

# **IBM uDeploy® User Guide**

**5.0.0**

---

# IBM uDeploy User Guide: 5.0.0

Publication date January 2013

Copyright © 2013 UrbanCode, an IBM Company, Inc.

UrbanCode, an IBM Company, AnthillPro, IBM uDeploy and any other product or service name or slogan or logo contained in this documentation are trademarks of UrbanCode, an IBM Company and its suppliers or licensors and may not be copied, imitated, or used, in whole or in part, without the prior written permission of UrbanCode, an IBM Company or the applicable trademark holder. Ownership of all such trademarks and the goodwill associated therewith remains with UrbanCode, an IBM Company or the applicable trademark holder.

Reference to any products, services, processes, or other information, by trade name, trademark, or otherwise does not constitute or imply endorsement, sponsorship, or recommendation thereof by UrbanCode, an IBM Company.

All other marks and logos found in this documentation are the property of their respective owners. For a detailed list of all third party intellectual property mentioned in our product documentation, please visit: <http://www.UrbanCode, an IBM Company.com/html/company/legal/trademarks.html>.

Document Number: 5.0.0.1u

---

---

About This Book .....	1
Product Documentation .....	1
Product Support .....	1
Document Conventions .....	1
Getting Started .....	3
IBM uDeploy Roadmap .....	4
Installing and Upgrading Servers and Agents .....	7
Installation Recommendations .....	8
System Requirements .....	8
Server Minimum Installation Requirements .....	8
Recommended Server Installation .....	8
Agent Minimum Requirements .....	9
32- and 64-bit JVM Support .....	9
Performance Recommendations .....	10
Download IBM uDeploy .....	10
Database Installation .....	11
Installing Oracle .....	11
Installing MySQL .....	11
Installing Microsoft SQL Server .....	12
Server Installation .....	13
Windows Server Installation .....	13
Unix/Linux Installation .....	15
Agent Installation .....	16
Installing an Agent .....	17
Connecting Agents to Agent Relays .....	18
Installing Agent Relays .....	18
Upgrading IBM uDeploy .....	20
Upgrading Agents .....	20
SSL Configuration .....	20
Configuring SSL Unauthenticated Mode for HTTP Communications .....	21
Configuring Mutual Authentication .....	22
Running IBM uDeploy .....	23
Running the Server .....	23
Running an Agent .....	23
Running an Agent Relay .....	23
Accessing IBM uDeploy .....	24
Quick Start—helloWorld Deployment .....	25
Creating Components .....	25
<i>helloWorld</i> Deployment .....	26
A Note Before You Begin .....	26
<i>helloWorld</i> Component Version .....	26
Component Process .....	29
<i>helloWorld</i> Process Design .....	30
<i>helloWorld</i> Application .....	35
Creating an Application .....	36
Adding the <i>helloWorld</i> Component to the Application .....	36
Adding an Environment to the Application .....	36
Adding a Process to the Application .....	38
Designing the Process Steps .....	38
Running the Application .....	40
Using IBM uDeploy .....	43
Components .....	44
Creating Components .....	44
Importing/Exporting Components .....	46

---

Component Properties .....	48
Component Versions .....	49
Importing Versions Manually .....	50
Importing Versions Automatically .....	51
Component Version Statuses .....	52
Deleting Component Versions .....	52
Inactivating Component Versions .....	52
Component Processes .....	52
Configuring Component Processes .....	52
Process Editor .....	54
To Display the Process Editor .....	54
Using the Process Editor .....	55
Adding Process Steps .....	56
Connecting Process Steps .....	58
Process Properties .....	59
Switch Steps and Conditional Processes .....	60
Process Step Properties .....	61
Component Manual Tasks .....	62
Creating Component Manual Tasks .....	62
Using Component Manual Tasks .....	62
Post-Processes .....	63
Component Templates .....	63
Creating a Component Template .....	63
Importing\Exporting Templates .....	64
Component Template Properties .....	65
Using Component Templates .....	67
Configuration Templates .....	67
Component Change Logs .....	68
Deleting and Deactivating Components .....	68
Resources .....	70
Resource Groups .....	70
Creating a Resource Group .....	70
Resource Roles .....	72
Role Properties .....	72
Agents .....	72
Remote Agent Installation .....	73
Managing Agents Remotely .....	74
Agent Pools .....	75
Creating an Agent Pool .....	75
Managing Agent Pools .....	75
Applications .....	76
Creating Applications .....	77
Adding Components to an Application .....	78
Importing/Exporting Applications .....	78
Application Environments .....	80
Creating an Environment .....	80
Mapping Resources to an Environment .....	81
Environment Properties .....	82
Application Processes .....	82
Creating Application Processes .....	83
Application Process Steps .....	84
Application Process Steps Details .....	84
Application Manual Tasks .....	87
Creating Application Manual Tasks .....	87

Using Manual Tasks .....	88
Approval Process .....	88
Work Items .....	88
Snapshots .....	89
Creating Snapshots .....	89
Application Gates .....	89
Creating Gates .....	90
Generic Processes .....	92
Deployments .....	93
Scheduling Deployments .....	96
Reports .....	97
Deployment Reports .....	97
Deployment Detail Report .....	98
Deployment Count Report .....	100
Deployment Average Duration Report .....	103
Deployment Total Duration Report .....	106
Security Reports .....	108
Application Security Report .....	108
Component Security Report .....	109
Environment Security Report .....	110
Resource Security Report .....	110
Saving and Printing Reports .....	111
Saving Report Data .....	111
Saving Report Filters .....	111
Printing Reports .....	112
Reference .....	113
Component Source Configuration .....	114
Basic Fields .....	114
File System (Basic and Versioned) .....	115
File System (Basic) .....	115
File System (Versioned) .....	115
Serena Dimensions CM .....	116
Plug-ins .....	117
Standard Plug-ins .....	118
Creating Plug-ins .....	118
The plugin.xml File .....	119
Plug-in Steps--the <step-type> Element .....	121
The <command> Element .....	122
The <post-processing> Element .....	123
Upgrading Plug-ins .....	125
The info.xml File .....	125
Example Plug-in .....	125
Step Properties .....	127
Step Commands .....	129
The <post-processing> Element .....	131
IBM uDeploy Properties .....	132
Command Line Client (CLI) Reference .....	136
Command Format .....	136
Commands .....	137
addActionToRoleForApplications .....	137
addActionToRoleForComponents .....	137
addActionToRoleForEnvironments .....	138
addActionToRoleForResources .....	138
addActionToRoleForUI .....	138

addAgentToPool .....	139
addComponentToApplication .....	139
addGroupToRoleForApplication .....	140
addGroupToRoleForComponent .....	140
addGroupToRoleForEnvironment .....	141
addGroupToRoleForResource .....	141
addGroupToRoleForUI .....	142
addLicense .....	142
addNameConditionToGroup .....	142
addPropertyConditionToGroup .....	143
addResourceToGroup .....	143
addRoleToResource .....	144
addRoleToResourceWithProperties .....	144
addUserToGroup .....	145
addUserToRoleForApplication .....	145
addUserToRoleForComponent .....	145
addUserToRoleForEnvironment .....	146
addUserToRoleForResource .....	146
addUserToRoleForUI .....	147
addVersionFiles .....	147
addVersionStatus .....	148
createAgentPool .....	148
createApplication .....	149
createApplicationProcess .....	149
createComponent .....	150
createComponentProcess .....	150
createDynamicResourceGroup .....	151
createEnvironment .....	151
createGroup .....	152
createMapping .....	152
createResource .....	152
createResourceGroup .....	153
createRoleForApplications .....	153
createRoleForComponents .....	154
createRoleForEnvironments .....	154
createRoleForResources .....	155
createRoleForUI .....	155
createSubresource .....	155
createUser .....	156
createVersion .....	156
deleteAgent .....	157
deleteAgentPool .....	157
deleteGroup .....	157
deleteResource .....	158
deleteResourceGroup .....	158
deleteResourceProperty .....	158
deleteUser .....	159
exportGroup .....	159
getAgent .....	159
getAgentPool .....	160
getAgentPools .....	160
getAgents .....	160
getApplication .....	161
getApplicationProcess .....	161

getApplicationProcessRequestStatus .....	161
getApplicationProperties .....	162
getApplicationProperty .....	162
getApplications .....	162
getComponent .....	163
getComponentEnvironmentProperties .....	163
getComponentEnvironmentProperty .....	163
getComponentProcess .....	164
getComponents .....	164
getComponentsInApplication .....	165
getComponentProperties .....	165
getComponentProperty .....	165
getEnvironment .....	166
getEnvironmentProperties .....	166
getEnvironmentProperty .....	166
getEnvironmentsInApplication .....	167
getGroupsForResource .....	167
getMapping .....	168
getMappingsForApplicationEnvironment .....	168
getMappingsForGroup .....	168
getResource .....	169
getResourceGroup .....	169
getResourceGroups .....	169
getResourceProperties .....	170
getResourceProperty .....	170
getResources .....	170
getResourcesInGroup .....	171
getResourceSecurity .....	171
getRoleForApplications .....	171
getRoleForComponents .....	172
getRoleForEnvironments .....	172
getRoleForResources .....	172
getRoleForUI .....	173
getRolesForResource .....	173
getSystemProperties .....	173
getSystemProperty .....	174
getUser .....	174
importGroup .....	174
importVersions .....	175
inactivateEnvironment .....	175
installAgent .....	176
login .....	177
logout .....	177
removeActionFromRoleForApplications .....	177
removeActionFromRoleForComponents .....	178
removeActionFromRoleForEnvironments .....	178
removeActionFromRoleForResources .....	179
removeActionFromRoleForUI .....	179
removeAgentFromPool .....	179
removeGroupFromRoleForApplication .....	180
removeGroupFromRoleForComponent .....	180
removeGroupFromRoleForEnvironment .....	181
removeGroupFromRoleForResource .....	181
removeGroupFromRoleForUI .....	182

removeMapping .....	182
removeResourceFromGroup .....	183
removeRoleForApplications .....	183
removeRoleForComponents .....	183
removeRoleForEnvironments .....	184
removeRoleForResources .....	184
removeRoleForUI .....	185
removeRoleFromResource .....	185
removeUserFromGroup .....	185
removeUserFromRoleForApplication .....	186
removeUserFromRoleForComponent .....	186
removeUserFromRoleForEnvironment .....	187
removeUserFromRoleForResource .....	187
removeUserFromRoleForUI .....	188
removeVersionStatus .....	188
repeatApplicationProcessRequest .....	189
requestApplicationProcess .....	189
restartAgent .....	189
setApplicationProperty .....	190
setComponentProperty .....	190
setComponentEnvironmentProperty .....	191
setEnvironmentProperty .....	191
setResourceProperty .....	192
setSystemProperty .....	192
shutdownAgent .....	193
testAgent .....	193
updateUser .....	194
Best Practices .....	195
General Practices .....	195
Maintenance .....	197
Rollbacks .....	198
Security .....	198
Glossary .....	201
Index .....	211



---

# About This Book

This book describes how to use UrbanCode, an IBM Company's IBM uDeploy product and is intended for all users.

## Product Documentation

IBM uDeploy's online Help is installed along with the product software and can be accessed from the product's web-based user interface. Product books are available in PDF format at UrbanCode, an IBM Company's Documentation portal: <http://docs.urbancode.com/>, and are included with the installation package.

In addition to this book, the books comprising the documentation suite include:

**Table 1. Product Documentation Suite**

Book	Description
IBM uDeploy User Guide	Describes how to use the product; contains chapters for core features, such as components, applications, and resources.
IBM uDeploy Administration Guide	Describes how to install and configure the product, and how to set up security.
IBM uDeploy Introduction	Provides an overview of the product's significant features and describes its architecture.

## Product Support

The UrbanCode, an IBM Company Support portal, <http://support.urbancode.com/>, provides information that can address any of your questions about the product. The portal enables you to:

- review product FAQs
- download patches
- view release notes that contain last-minute product information
- review product availability and compatibility information
- access white papers and product demonstrations

## Document Conventions

This book uses the following special conventions:

- Program listings, code fragments, and literal examples are presented in this typeface.
- Product navigation instructions are provided like this:

*Home > Components > [selected component] > Versions > [selected version] > Add a Status [button]*

This example, which explains how to add a status to a component version, means: from the IBM uDeploy home page click the Components tab (which displays the Components pane); select a component (which displays a pane with information for the selected component); click the Versions tab (which displays a pane with information about the selected version); and click the Add a Status button.

- User interface objects, such as field and button names, are displayed with initial Capital Letters.
- Variable text in path names or user interface objects is displayed in *italic* text.
- Information you are supposed to enter is displayed in this format.

---

# Getting Started

---

---

# IBM uDeploy Roadmap

This chapter provides information that will help you quickly become productive with IBM uDeploy. It provides the "happy path" to productivity: it describes how to create components and define applications to deploy them, and, finally, describes how to perform deployments. The following topics should be reviewed in order.

- Creating components
- Creating applications
- Deploying components

Other topics you might find of interest are provided in the section called "Other Topics". This book also provides a step-by-step tutorial on creating components and applications.

## Create a Component

Components are the centerpiece of IBM uDeploy's deployment engine. Components associate items that will be deployed—artifacts—with processes that will deploy them. The following table summarizes the basic steps performed to create components. Related topics are listed below the table.

**Table 2. Component Creation Steps**

Step	Description
1. Define source configuration	Define the source type and identify the artifacts associated with the component. The source type can be any or nearly any associated with a software project. Once defined, all artifacts must be of the defined type. See the section called "Creating Components".
2. Create component version	Create the initial component version by importing artifacts into the artifact repository, CodeStation. Versions can be imported manually or automatically. Version imports can be full (all artifacts are imported) or incremental (only changed artifacts are imported). IBM uDeploy tracks all artifact changes which enables you to rollback components or deploy multiple versions of the same one.
3. Create component process	Use the process design editor to create a process for the component. Component processes consist of user-configured steps that operate on the component, usually by deploying it. The available steps are provided by installed plug-ins. As shipped IBM uDeploy provides plug-ins for many common functions. Numerous other plug-ins are available from UrbanCode— <a href="http://plugins.urbancode.com">http://plugins.urbancode.com</a> .

Related topics:

- How to create manual tasks

- How to install plug-ins (see the IBM uDeploy Administration Guide)
- How to create and use templates
- How to import components

## Create an Application

Applications associate components with the agents that will manage them, and define processes to perform deployments.

The following table summarizes the steps performed to create applications.

**Table 3. Application Creation Steps**

Step	Description
1. Create an application and identify its components	After defining the application, identify the components it will manage. Associating a component makes its processes and properties available to the application. An application can have any number of components associated with it.
2. Create an environment	Define an environment and use it to map an agent to component(s). Mapping means assigning an agent to manage the component. Each component can be mapped to the same agent, a different one, or some combination. An application can have more than one environment defined for it.
3. Create an application process	Use the process design editor to create a process. Application processes are created with the same editor used to create the component process, but uses a different toolkit of process steps. Previously defined component processes can be incorporated into the process.

Related topics:

- Learn about IBM uDeploy properties
- How to create snapshots
- How to import applications

## Deploy the Component

Components are deployed by application processes. The following table summarizes the steps performed to run an application process.

**Table 4. Deployment Steps**

Step	Description
1. Select environment	Application processes are run at the environment level; you run a process for a particular environment. Selecting an environment automatically selects its agent(s). All processes defined for the application are available.

Step	Description
2. Run processs	You run a process by selecting it for a given environment and specifying certain other parameters. Processes can also be run with the CLI, or scheduled for a future time.
3. Check results	When a process is started, the Application Process Request pane displays information about the application's status and provides links to logs and the application manifest. If an approval or manual task was used, this pane enables affected users to respond.

Related topics:

- How to create notification schemes (see the IBM uDeploy Administration Guide)
- How to setup authorizations
- How to create application gates

## Other Topics

The following list provides links to additional topics.

- How to run reports
- How to use the command line interface
- How to create plug-ins
- How to add agents to a product license (see the IBM uDeploy Administration Guide)

---

# Installing and Upgrading Servers and Agents

A IBM uDeploy installation consists of the IBM uDeploy server (with a supporting database), and at least one agent. Typically, the server, database, and agents are installed on separate machines, although for a simple evaluation they can all be installed on the same machine. In addition, Java must be installed on all agent and server machines.

## Note

For evaluation purposes, the supplied Derby database should be adequate and can be installed on the machine where the server is located. If you are installing IBM uDeploy in a production environment, UrbanCode, an IBM Company recommends the use one of the supported databases--Oracle Database (all versions), SQL Server, or MySQL.

## Installation Steps

1. Review the system requirements. See the section called “System Requirements”. Requirements and recommendations, including performance recommendations, are provided.
2. Ensure that Java is installed on the server and agent machines (and agent relay machine if used). All machines require Java JRE 5 or greater. Set the JAVA\_HOME environment variable to point to the directory you intend to use. A JDK can be used.
3. Download the server, agent, agent relay, and CLI client (command line interface) installation packages. Installation files can be downloaded from the UrbanCode support portal <http://support.urbancode.com>. If you are installing an evaluation version, the license is included with the downloaded files. For evaluations, the agent relay (used to communicate with remote networks) and the CLI client can be skipped. At a minimum, an installation must have the server, a database, and at least one agent.
4. If you are installing an agent relay, download the agent relay installation files as well.
5. If you are not installing an evaluation version, install one of the supported databases. The database should be installed before the server and on a separate machine. See the section called “Database Installation”
6. Complete database installation by configuring the appropriate JDBC driver (typically supplied by the database vendor).
7. Create an empty database for IBM uDeploy and at least one dedicated user account.
8. Install the server. You will need to supply values for the IP address, ports for HTTP communication (secured and unsecured), port for agent communication, and URL. The installation program provides default values for many parameters. The properties set during installation are recorded in the `installed.properties` file located in the `server_install/conf/server/` directory. If you intend to turn on SSL, see the section called “SSL Configuration” See the section called “Server Installation”.
9. If you are using an agent relay, install the relay. See the section called “Installing Agent Relays”.
10. Finally, install at least one agent. Agents are installed on target machines and communicate with the server. When installing an agent, you supply several values defined during server installation. See the section called “Agent Installation” for instructions about installing agents. An agent requires various

access privileges for the machine where it is installed, which are described in that section. To determine if the agent is in communication with the server, display the web application's Resource pane. A value of *Online* in the agent's Status field means the agent is successfully connected.

For information about using the CLI (command line interface, see *Command Line Client (CLI) Reference*.

For information about running the installed items and accessing the IBM uDeploy web application, see the section called “Running IBM uDeploy”.

## Installation Recommendations

Because the IBM uDeploy agent performs most of the deployment processing, agent installation is critical for good performance. Except for evaluation purposes, an agent should never be installed on the same machine as the server. In addition, many IBM uDeploy users have found that by following some general guidelines they are able to reduce the chances of performance-related issues:

- **Install the server as a user account.** The server should be installed as a dedicated system account whenever possible. While not recommended, IBM uDeploy can run as the root user (or local system user on Windows) and running in this manner avoids all permission errors.
- **Install each agent as a dedicated system account.** Ideally, the account should only be used by IBM uDeploy. Because IBM uDeploy agents are command execution engines, it is advisable to limit what they can do on host machines by creating dedicated users and then granting them appropriate privileges. If you install an agent as the root user (or local system user on Windows), ensure that agent processes cannot adversely effect the host file system.
- **Except for evaluation purposes, do not install an agent on the IBM uDeploy server machine.** Because the agent is resource intensive, installing one on the server machine can degrade performance.
- **Install a single agent per host machine.** Multiple agents on the same machine can negatively impact each other's performance. When you must install multiple agents, you might see performance degradation when multiple agents are busy simultaneously.

## System Requirements

IBM uDeploy will run on Windows and Unix-based systems. While the minimum requirements provided below are sufficient for an evaluation, you will want server-class machines for production deployments.

### Server Minimum Installation Requirements

- Windows: Windows 2000 Server (SP4) or later.
- Processor: Single core, 1.5 GHz or better.
- Disk Space: 300 MB or more.
- Memory: 2 GB, with 256 MB available to IBM uDeploy.
- Java version: JRE 5 or greater.

### Recommended Server Installation

- **Two server-class machines**



UrbanCode, an IBM Company recommends two machines for the server: a primary machine and a standby for fail-over. In addition, the database should be hosted on a separate machine.

- **Separate machine for the database**
- **Processor** 2 CPUs, 2+ cores for each.
- **RAM** 8 GB
- **Storage** Individual requirements depend on usage, retention policies, and application types. In general, the larger number of artifacts kept in IBM uDeploy's artifact repository (CodeStation), the more storage needed.

## Note

CodeStation is installed when the IBM uDeploy server is installed.

For production environments, use the following guidelines to determine storage requirements:

- 10-20 GB of database storage should be sufficient for most environments.
- To calculate CodeStation storage requirements:

*average artifact size \* number of versions imported per day \* average number of days before cleanup*

- Approximately 1MB per deployment of database storage; varies based on local requirements.

For further assistance in determining storage requirements, contact UrbanCode, an IBM Company support.

- **Network** Gigabit (1000) Ethernet with low-latency to the database.

## Agent Minimum Requirements

Designed to be minimally intrusive (typically, an idle agent uses 5Mz CPU), agents require 64-256 MB of memory and 100 MB of disk space. Additional requirements are determined by the processes the agent will run. For a simple evaluation, the agent can be installed on the same physical machine as the server. In production environments, agents should be installed on separate machines.

## 32- and 64-bit JVM Support

The IBM uDeploy server must use the 32-bit JDK for the Windows 2003 64-bit server; the 64-bit JDK can be used for agents. Because IBM uDeploy does not require a multi-gigabyte heap, there is little advantage to using a 64-bit JVM. For 64-bit Windows installations, IBM uDeploy uses a 32-bit JVM; for other 64-bit platforms, IBM uDeploy uses a 64-bit JVM, as the following table illustrates:

**Table 5. JVM Support**

Operating System	Operating System	JVM 64-bit
Windows 32-bit	yes	NA
Windows 64-bit	yes	yes

Operating System	Operating System	JVM 64-bit
Non-Windows 32-bit	yes	NA
Non-Windows 64-bit	yes	yes

## Performance Recommendations

Since the IBM uDeploy agent performs most of the processing, optimal agent configuration is important. Except when evaluating IBM uDeploy, an agent should not be installed on the same machine where the server is located.

By following these recommendations, you should avoid most performance-related issues:

- **Install the server as a dedicated user account.** The server should be installed as a dedicated system account whenever possible. However, IBM uDeploy runs well as a root user (or local system user on Windows), and running this way is the easiest method to avoid permission errors.
- **Install the agent as dedicated system account.** Ideally, the account used should be dedicated to IBM uDeploy. Because IBM uDeploy agents are remote command-execution engines, it is best to create a user just for the agent and grant it only the appropriate privileges.
- **Do not install an agent on the IBM uDeploy server machine.** Because the agent is resource intensive, installing one on the server machine will degrade server performance whenever a large deployment runs.
- **Install one agent per machine.** Several agents on the same machine can result in significant performance reduction, especially when they are running at the same time.

## Download IBM uDeploy

The installation package is available from the UrbanCode, an IBM Company support portal--Supportal. If you are evaluating IBM uDeploy, the Supportal account where you download IBM uDeploy also enables you to create support tickets.

1. Navigate to the UrbanCode, an IBM Company Support Portal *support.UrbanCode, an IBM Company.com/tasks/login/LoginTasks/login*. If you do not have an account, please create one.

### Note

You must have a license in order to download the product. For an evaluation license, go to *UrbanCode, an IBM Company.com/html/products/deploy/default.html*.

2. Click the **Products** tab and select the IBM uDeploy version you want to download.
3. Select the appropriate package for your environment for the server, agent, command line client, and agent relay. The contents of the zip and tar packages are the same.

IBM uDeploy enables you to install agents on any supported platform, regardless of the operating system where the server is installed.

4. Download the license. If you do not see a license, ensure that you are the Supportal account administrator. Licenses are not available to all Supportal users.

# Database Installation

Currently, IBM uDeploy supports Derby, Oracle, SQL Server, and MySQL.

## Installing Oracle

Before installing the IBM uDeploy server, install an Oracle database. If you are evaluating IBM uDeploy, you can install the database on the same machine where the IBM uDeploy server will be installed.

When you install IBM uDeploy, you will need the Oracle connection information, and a user account with table creation privileges.

### IBM uDeploy supports the following editions:

- Oracle Database Enterprise
- Oracle Database Standard
- Oracle Database Standard One
- Oracle Database Express

Version 10g or later is supported for each edition.

### To install the database

1. Obtain the Oracle JDBC driver. The JDBC jar file is included among the Oracle installation files. The driver is unique to the edition you are using.
2. Copy the JDBC jar file to *IBM uDeploy\_installer\_directory\lib\ext*.
3. Begin server installation, see the section called “Server Installation”. When you are prompted for the database type, enter *oracle*.
4. Provide the JDBC driver class IBM uDeploy will use to connect to the database.

The default value is *oracle.jdbc.driver.OracleDriver*.

5. Provide the JDBC connection string. The format depends on the JDBC driver. Typically, it is similar to:

```
jdbc:oracle:thin:@[DB_URL]:[DB_PORT]
```

For example:

```
jdbc:oracle:thin:@localhost:1521.
```

6. Finish by entering the database user name and password.

### Note

The schema name must be the same as the user name.

## Installing MySQL

Before installing the IBM uDeploy server, install MySQL. If you are evaluating IBM uDeploy, you can install the database on the same machine where the IBM uDeploy server will be installed.

When you install IBM uDeploy, you will need the MySQL connection information, and a user account with table creation privileges.

### To install the database

1. Create a database:

```
CREATE DATABASE IBM uDeploy;  
  
GRANT ALL ON IBM uDeploy * TO 'IBM uDeploy'@'%'  
  
IDENTIFIED BY 'password' WITH GRANT OPTION;
```

2. Obtain the MySQL JDBC driver. The JDBC jar file is included among the installation files. The driver is unique to the edition you are using.
3. Copy the JDBC jar file to *IBM uDeploy\_installer\_directory*\lib\ext.
4. Begin server installation, see the section called “Server Installation”. When you are prompted for the database type, enter *mysql*.
5. Provide the JDBC driver class IBM uDeploy will use to connect to the database.

The default value is *com.mysql.Driver*.

6. Next, provide the JDBC connection string. Typically, it is similar to:

```
jdbc:mysql[DB_URL]:[DB_PORT]:[DB_NAME]
```

For example:

```
jdbc:mysql://localhost:3306/IBM uDeploy.
```

7. Finish by entering the database user name and password.

## Installing Microsoft SQL Server

Before installing the IBM uDeploy server, install a SQL Server database. If you are evaluating IBM uDeploy, you can install the database on the same machine where the IBM uDeploy server will be installed.

When you install IBM uDeploy, you will need the SQL Server connection information, and a user account with table creation privileges.

Before installing the IBM uDeploy server, install an SQL Server database. If you are evaluating IBM uDeploy, you can install the database on the same machine where the IBM uDeploy server will be installed:

```
CREATE DATABASE IBM uDeploy;  
  
USE IBM uDeploy;  
  
CREATE LOGIN IBM uDeploy WITH PASSWORD = 'password';  
  
CREATE USER IBM uDeploy FOR LOGIN IBM uDeploy WITH DEFAULT_SCHEMA = IBM uDeploy;  
  
CREATE SCHEMA IBM uDeploy AUTHORIZATION IBM uDeploy;
```

```
GRANT ALL TO IBM uDeploy;
```

1. Obtain the SQL Server JDBC driver from the Microsoft site. The JDBC jar file is not included among the installation files.
2. Copy the JDBC jar file to *IBM uDeploy\_installer\_directory\lib\ext*.
3. Begin server installation, see the section called “Server Installation”. When you are prompted for the database type, enter *sqlserver*.
4. Provide the JDBC driver class IBM uDeploy will use to connect to the database.

The default value is *com.microsoft.sqlserver.jdbc.SQLServerDriver*.

5. Next, provide the JDBC connection string. The format depends on the JDBC driver. Typically, it is similar to:

```
jdbc:sqlserver://[DB_URL]:[DB_PORT];databaseName=[DB_NAME]
```

For example:

```
jdbc:sqlserver://localhost:1433;databaseName=IBM uDeploy.
```

6. Finish by entering the database user name and password.

## Server Installation

The server provides services such as the user interface used to configure application deployments, the work flow engine, the security service, and the artifact repository, among others. The properties set during installation are recorded in the *installed.properties* file located in the *server\_install/conf/server/* directory.

If the following steps fail, contact UrbanCode support and provide the log from standard out put.

### Note

If you are installing the server in a production environment, install and configure the database you intend to use before installing the server. See the section called “Database Installation”.

## Windows Server Installation

1. Download and unpack the installation files to the *installer\_directory*.
2. From the *installer\_directory*, run *install-server.bat*.

### Note

Depending on your Windows version, you might need to run the batch file as the administrator.

The IBM uDeploy Installer is displayed and prompts you to provide the following information:

3. **Enter the directory where the IBM uDeploy Server will be installed.** Enter the directory where you want the server located. If the directory does not exist, enter *Y* to instruct the Installer to create it for you. If you enter an existing directory, the program will give you the option to upgrade the server. For information about upgrading, see the section called “Upgrading IBM uDeploy”.

## Note

Any default values suggested by the program (displayed within brackets) can be accepted by simply pressing **Enter**. If two options are given, such as Y/n, the capitalized option is the default value.

**4. Please enter the home directory of the JRE/JDK used to run the server.**

If Java has been previously installed, IBM uDeploy will suggest the Java location as the default value. To accept the default value, press *ENTER*, otherwise override the default value and enter the correct path.

**5. Enter the IP address on which the Web UI should listen.** UrbanCode, an IBM Company suggests accepting the default value *all available to this machine*.

**6. Do you want the Web UI to always use secure connections using SSL?**

Default value is Y.

If you use SSL, turn it on for agents too, or the agents will not be able to connect to the server. This also applies if using mutual authentication. If you change the port numbers for agent communication, you need to provide the port numbers when installing the agents.

This sets the `install.server.web.always.secure=` property in the `installed.properties` file.

**7. Enter the port where IBM uDeploy should listen for secure HTTPS requests.** The default value is *8443*.

This sets the `install.server.web.ip=` property in the `installed.properties` file.

**8. Enter the port on which the IBM uDeploy server should redirect unsecured HTTP requests.**

The default value is *8080*.

**9. Enter the URL for external access to the web UI.**

**10. Enter the port to use for agent communication.**

The default value is *7918*.

**11. Do you want the Server and Agent communication to require mutual authentication?**

If you select Y, a manual key must be exchanged between the server and each agent. The default value is N.

This sets the `server.jms.mutualAuth=` property in the `installed.properties` file.

**12. Enter the database type IBM uDeploy should use.**

The default value is the supplied database *Derby*. The other supported databases are: *mysql*, *oracle*, and *sqlserver*.

If you enter a value other than *derby*, the IBM uDeploy Installer will prompt you for connection information, which was defined when you installed the database. See the section called “Database Installation”.

13. **Enter the database user name..** The default value is *IBM uDeploy*. Enter the user name you created during database installation.

14. **Enter the database password..** The default value is *password*.

15. **Do you want to install the Server as Windows service?** The default value is *N*.

When installed as a service, IBM uDeploy only captures the value for the PATH variable. Values captured during installation will always be used, even if you make changes later. For recent Windows versions, you will need to execute the command as Administrator.

### Note

If you install the server as a service, the user account must have the following privileges:

- SE\_INCREASE\_QUOTA\_NAME "Adjust memory quotas for a process"
- SE\_ASSIGNPRIMARYTOKEN\_NAME "Replace a process level token"
- SE\_INTERACTIVE\_LOGON\_NAME "Logon locally"

The LOCAL SYSTEM account is on every Windows machine and automatically has these privileges. You might want to use it as it requires minimal configuration.

## Unix/Linux Installation

1. Download and unpack the installation files to the *installer\_directory*.

### Note

If you are installing IBM uDeploy on Solaris, UrbanCode, an IBM Company recommends the Korn shell (ksh).

2. From the *installer\_directory* run *install-server.sh*. The IBM uDeploy Installer is displayed and prompts you to provide the following information:
3. **Enter the directory where the IBM uDeploy Server will be installed.** If the directory does not exist, enter *Y* to instruct the Installer to create it for you. The default value is *Y*.

### Note

Whenever the IBM uDeploy Installer suggests a default value, you can press *ENTER* to accept the value.

4. **Please enter the home directory of the JRE/JDK used to run the server.**

If Java has been previously installed, IBM uDeploy will suggest the Java location as the default value. To accept the default value, press *ENTER*, otherwise override the default value and enter the correct path.

5. **Enter the IP address on which the Web UI should listen.** UrbanCode, an IBM Company suggests accepting the default value *all available to this machine*.
6. **Do you want the Web UI to always use secure connections using SSL?**

Default value is *Y*.

If you use SSL, turn it on for agents too, or the agents will not be able to connect to the server. This also applies if using mutual authentication. If you change the port numbers for agent communication, you need to provide the port numbers when installing the agents.

This sets the `install.server.web.always.secure=` property in the `installed.properties` file.

7. **Enter the port where IBM uDeploy should listen for secure HTTPS requests.** The default value is *8443*.

This sets the `install.server.web.ip=` property in the `installed.properties` file.

8. **Enter the port on which the IBM uDeploy should redirect unsecured HTTP requests.**

The default value is *8080*.

9. **Enter the URL for external access to the web UI.**

10. **Enter the port to use for agent communication.**

The default value is *7918*.

11. **Do you want the Server and Agent communication to require mutual authentication?**

If you select *Y*, a manual key must be exchanged between the server and each agent. The default value is *N*.

This sets the `server.jms.mutualAuth=` property in the `installed.properties` file.

12. **Enter the database type IBM uDeploy should use.**

The default value is the supplied database *Derby*. The other supported databases are: *mysql*, *oracle*, and *sqlserver*.

If you enter a value other than *derby*, the IBM uDeploy Installer will prompt you for connection information, which was defined when you installed the database. See the section called “Database Installation”.

13. **Enter the database user name..** The default value is *IBM uDeploy*. Enter the user name you created when you installed the database.

14. **Enter the database password..** The default value is *password*.

## Agent Installation

For production environments, UrbanCode, an IBM Company recommends creating a user account dedicated to running the agent on the machine where the agent is installed.

For simple evaluations, the administrative user can run the agent on the machine where the server is located. But if you plan to run deployments on several machines, a separate agent should be installed on each machine. If, for example, your testing environment consists of three machines, install an agent on each one. Follow the same procedure for each environment the application uses.

Each agent needs the appropriate rights to communicate with the IBM uDeploy server (if the agent will communicate with IBM uDeploy via an agent relay, see the section called “Connecting Agents to Agent Relays”).



At a minimum, each agent should have permission to:

- **Create a cache.** By default, the cache is located in the home directory of the user running the agent. The cache can be moved or disabled.
- **Open a TCP connection.** The agent uses a TCP connection to communicate with the server's JMS port.
- **Open a HTTP(S) connection.** The agent must be able to connect to the IBM uDeploy user interface in order to download artifacts from the CodeStation repository.
- **Access the file system.** Many agents need read/write permissions to items on the file system.

## Installing an Agent

After downloading and expanding the installation package, open the *installer\_directory*.

From the *installer\_directory* run *install-agent.bat* (Windows) or *install-agent.sh* (Unix-Linux).

### Note

If you install the agent as a Windows service, the user account must have the following privileges:

- SE\_INCREASE\_QUOTA\_NAME "Adjust memory quotas for a process"
- SE\_ASSIGNPRIMARYTOKEN\_NAME "Replace a process level token"
- SE\_INTERACTIVE\_LOGON\_NAME "Logon locally"

The LOCAL SYSTEM account is on every Windows machine and automatically has these privileges. You might want to use it as it requires minimal configuration.

The IBM uDeploy Installer is displayed and prompts you to provide the following information. Any default values suggested by the program (displayed within brackets) can be accepted by simply pressing **Enter**. If two options are given, such as Y/n, the capitalized option is the default value.

1. **Enter the directory where agent should be installed..** For example: C:\Program Files\urban-deploy\agent (Windows) or /opt/urban-deploy/agent (Unix). If the directory does not exist, enter Y to instruct the Installer to create it for you. If you enter an existing directory, the program will give you the option to upgrade the agent. For information about upgrading, see the section called "Upgrading IBM uDeploy".

### Note

Any default values suggested by the program (displayed within brackets) can be accepted by simply pressing **Enter**. If two options are given, such as Y/n, the capitalized option is the default value.

2. **Please enter the home directory of the JRE/JDK used to run the agent.** If Java has been previously installed, IBM uDeploy will suggest the Java location as the default value. To accept the default value, press **ENTER**, otherwise override the default value and enter the correct path.
3. **Will the agent connect to a agent relay instead of directly to the server?** The default value is N. If the agent will connect to an agent relay, see the section called "Connecting Agents to Agent Relays".
4. **Enter the host name or address of the server the agent will connect to.** The default value is *localhost*.

5. **Enter the agent communication port for the server.** The default value is *7918*.

6. **Does the server agent communication use mutual authentication with SSL?** Default value is *Y*.

If you use SSL, turn it on for server too or the agent will not be able to connect to the server. This also applies if using mutual authentication. If you change the port numbers for agent communication, you need to provide them when installing the agents.

7. **Enter the name for this Agent.** Enter a unique name; the name will be used by IBM uDeploy to identify this agent. Names are limited to 256 characters and cannot be changed once connected.

8. **Do you want to install the Agent as Windows service?** (Windows only). The default value is *N*. When installed as a service, IBM uDeploy only captures the value for the PATH variable. Values captured during installation will always be used, even if you make changes later. For recent Windows versions, you will need to execute the command as Administrator.

Agents that will run on Unix machines can also be installed directly from the IBM uDeploy web application, see the section called “Agents”

### Note

If the agent is configured properly, IBM uDeploy will recognize it automatically—you do not need to perform further actions in order to start using it.

## Connecting Agents to Agent Relays

Remote agents--agents that will communicate with the server via an agent relay--are installed in much the same way local agents are installed (see the section called “Agent Installation”): you run the install script, *install-agent.bat*, and supply agent configuration information as described above, along with the relay-specific parameters.

When you answer *Yes* when asked if you want to connect the agent to a agent relay, you will be prompted to configure the following parameters:

**Table 6. Agent-Agent Relay Connection**

Parameter	Description
<b>hostname or address of the agent relay the agent will connect to</b>	Enter the host name or IP address of the agent relay. Supply the value you used when you installed the agent relay.
<b>agent communication port for the agent relay</b>	Enter the port which the agent will use for JMS-based communications with agent relay. The default value is 7916.
<b>HTTP proxy port for the agent relay</b>	Enter the port on which the agent will use for HTTP communications with the agent relay. The default value is 20080.

## Installing Agent Relays

An agent relay is a communication proxy for agents that are located behind a firewall or in another network location. As long as there is at least a low bandwidth WAN connection between the server and remote agents, the IBM uDeploy server can send work to agents located in other geographic locations via the

relay. An agent relay requires that only a single machine in the remote network contact the server. Other remote agents communicate with the server by using the agent relay. All agent-server communication from the remote network goes through the relay.

You can download the agent relay installation package from the UrbanCode, an IBM Company support portal--Supportal. Before installing, ensure that:

- Java 1.5.0 or later is installed.
- The server with which the relay will connect is already installed.
- The user account and password created during server installation is available.

**To install an agent relay:**

1. Expand the compressed installation file.
2. From within the expanded `agent-relay-install` directory run the `install.cmd` script.
3. The installation program will prompt you for the following information. Any default values suggested by the program (displayed within brackets) can be accepted by simply pressing **Enter**. If two options are given, such as Y/n, the capitalized option is the default value.

**Table 7. Agent Relay Configuration**

Parameter	Description
<b>Directory where you would like to install the agent relay</b>	Enter the directory where you want the agent relay installed. If you enter an existing directory, the program will prompt you to upgrade the relay. For information about upgrading, see the section called "Upgrading IBM uDeploy".
<b>Java home</b>	Directory where Java is installed. Ensure that the JAVA_HOME environment variable points to this directory.
<b>Name of this relay</b>	Enter the name of the agent relay. Each relay must have a unique name. The default name is <code>agent-relay</code> .
<b>IP or hostname which this agent relay should use</b>	Enter the IP or hostname on which the relay will listen.
<b>Port which this agent relay should proxy HTTP requests on</b>	Enter the port on which the agent relay should listen for HTTP requests coming from agents. The default value is 20080.
<b>Port which this agent relay should use for communication.</b>	Enter the port on which the agent relay will use for JMS-based communications with remote agents. The default value is 7916.
<b>Connect the agent relay to a central server?</b>	Specify whether you want the relay to connect to the IBM uDeploy server.
<b>IP or hostname of your central server</b>	If you indicated that you want to connect the relay to the server, enter the IP or host name where the relay can contact the server.
<b>Port which the central server uses for communication</b>	If you indicated that you want to connect the relay to the server, enter the port the server uses

Parameter	Description
	to communicate with agents. The default value is 7918.
Use secure communication between the agent, relay and server?	<p>Specify whether you want to use SSL security for communications between server, relay, and remote agents. The default value is Y.</p> <p><b>Important</b></p> <p>To use the relay, you must answer yes. Answering yes activates SSL security for HTTP- and JMS-based communications. If you answer no, the relay <i>will not</i> be able to communicate with the server (which uses JMS for most communications).</p>
Use mutual authentication between the agent, relay and server.	If mutual authentication is required, enter Y. See the section called “SSL Configuration” for information about activating mutual authentication.
Install the Agent Relay as Windows service?	If you are installing the relay on Windows, you can install it as a Windows service. The default value is N.

If you need to modify the relay, you can edit these properties in the `agentrelay.properties` file located in the `relay_installation\conf` directory.

## Upgrading IBM uDeploy

You upgrade the IBM uDeploy server, agents, and agent relays independently. Before upgrading, download the appropriate installation package from the UrbanCode, an IBM Company support portal (upgrades for servers and agent relays are done with the same package used for installation), and uncompress it.

1. Run the installation script for the server or agent relay you want to upgrade. To upgrade the server, for example, run the `install-server` script.
2. When prompted for the location of the installation directory, enter the path to an existing installation. When you specify an existing installation, IBM uDeploy will ask if you want to upgrade the installation (instead of installing a new version). If you answer Yes, the script will lead you through the required steps. The upgrade steps are a subset of the installation steps. If you need information about the steps, see the section related to the item you are upgrading.

## Upgrading Agents

Agents can be upgraded individually or in batch. To upgrade an agent, display the Agents pane (*Resources > Agents*), then use the Upgrade action. To upgrade multiple agents, select the agents and use the Upgrade Selected button.

## SSL Configuration

SSL (Secure Socket Layer) technology enables clients and servers to communicate securely by encrypting all communications. Data are encrypted before being sent and decrypted by the recipient--communications cannot be deciphered or modified by third-parties.

IBM uDeploy enables the server to communicate with its agents using SSL in two modes: unauthenticated and mutual authentication. In unauthenticated mode, communication is encrypted but users do not have to authenticate or verify their credentials. IBM uDeploy automatically uses this mode for JMS-based server/agent communication (you cannot turn this off). SSL unauthenticated mode can also be used for HTTP communication. You can implement this mode for HTTP communication during server/agent/agent relay installation, or activate it afterward, as explained below.

## Important

IBM uDeploy automatically uses SSL in unauthenticated mode for JMS-based communications between the server and agents (JMS is IBM uDeploy's primary communication method). Because agent relays do not automatically activate SSL security, you must turn it on during relay installation or before attempting to connect to the relay. Without SSL security active, agent relays cannot communicate with the server or remote agents.

In mutual authentication mode, the server, local agents, and agent relays each provide a digital certificate to one another. A digital certificate is a cryptographically signed document intended to assure others about the identity of the certificate's owner. IBM uDeploy certificates are self-signed. When mutual authentication mode is active, IBM uDeploy uses it for JMS-based server, local agents, and agent relay communication.

To activate this mode, the IBM uDeploy server provides a digital certificate to each local agent and agent relay, and each local agent and agent relay provides one to the server. Agent relays, in addition to swapping certificates with the server, must swap certificates with the remote agents that will use the relay. Remote agents do not have to swap certificates with the server, just with the agent relay it will use to communicate with the server. This mode can be implemented during installation or activated afterward, as explained below

## Note

When using mutual authentication mode, you must turn it on for the server, agents, and agent relays, otherwise they will not be able to connect to one another--if one party uses mutual authentication mode, they all must use it.

# Configuring SSL Unauthenticated Mode for HTTP Communications

To activate unauthenticated mode for HTTP:

1. Open the `installed.properties` file which is located in the `server_install/conf/server` directory. The `installed.properties` file contains the properties that were set during installation.
2. Ensure that the `install.server.web.always.secure` property is set to `Y`.
3. Ensure that the `install.server.web.ip` property is set to the port the server should use for HTTPS requests.
4. Save the file and restart the server.

## Note

To complete unauthenticated mode for HTTP, contact UrbanCode, an IBM Company Support.

# Configuring Mutual Authentication

To use mutual authentication, the server and agents must exchange keys. You export the server key (as a certificate) and import it into the agent keystore, then reverse the process by exporting the agent key and importing it into the server keystore. When using an agent relay, the relay must swap certificates with the server and with the remote agents that will use the relay.

**Before exchanging keys, ensure that the following properties are set:**

1. The `server.jms.mutualAuth` property in the server's `installed.properties` file (located in the `server_install/conf/server` directory) is set to `true`.
2. For each agent, the `locked/agent.mutual_auth` property in the agent's `installed.properties` file (located in the `agent_install\conf\agent` directory) is set to `true`.
3. For each agent relay, the `agentrelay.jms_proxy.secure` property in the relay's `agentrelay.properties` file (located in the `relay_install\conf` directory) is set to `true`.
4. For each agent relay, the `agentrelay.jms_proxy.mutualAuth` property in the relay's `agentrelay.properties` file is set to `true`.

**To exchange keys:**

1. Open a shell and navigate to the server installation `conf` directory.
2. Run:

```
keytool -export -keystore server.keystore -storepass changeit  
-alias server -file server.crt
```

3. Copy the exported file (certificate) to the local agent/agent relay installation `conf` directory.
4. Import the file by running from within the agent's `conf` directory (or agent relay's `jms-relay` directory):

```
keytool -import -keystore ud.keystore -storepass changeit  
-alias server -file server.crt -keypass changeit -noprompt
```

You should see the Certificate was added to keystore message.

## Note

For agent relays, replace `ud.keystore` with the name of the relay's keystore--`agentrelay.keystore`

5. For each local agent or agent relay, export the key by running the following (change the name of the file argument to match the agent name):

```
keytool -export -keystore ud.keystore -storepass changeit
```

```
-alias ud_agent -file [agent_name].cert
```

You should see the Certificate stored in file (agent\_name.cert) message.

### Note

For agent relays, replace ud.keystore with the name of the relay's keystore--agentrelay.keystore

6. Copy the exported file to the server's conf directory.

7. From within the server's conf directory, import each certificate by running the following command (change the name of the file argument and alias to match the certificate's name):

```
keytool -import -keystore ud.keystore -storepass changeit  
-alias [agent_name] -file [agent_name].cert -keypass changeit -noprompt
```

You should see the Certificate was added to keystore message.

8. Restart the server and agents/agent relays.

To connect an agent relay with the remote agents that will use it, swap certificates as explained above: each remote agent must import the certificate for the relay it will use, and the relay must import the certificate from each remote agent that will use it. Agents using relays do not have to swap certificates with the server.

To list the certificates loaded into a keystore, run the following from within the keystore directory:

```
keytool -list -keystore ud.keystore -storepass changeit
```

## Running IBM uDeploy

Both Unix- and Windows-based installations require the IBM uDeploy server and at least one agent. If you are using a Oracle or MySQL database, make sure you have installed and configured the appropriate driver, see the section called “Database Installation”.

### Running the Server

1. Navigate to the *server\_installation\bin* directory
2. Run the *run\_server.cmd* batch file (Windows), or *start\_server.cmd* (Unix/Linux).

### Running an Agent

After the server has successfully started:

1. Navigate to the *agent\_installation\bin* directory
2. Run the *run\_udagent.cmd* batch file (Windows), or *start\_udagent.cmd* (Unix/Linux).
3. Once the agent has started, navigate to the IBM uDeploy web application and display the **Resources** tab. If installation went well, the agent should be listed with a status of Online.

### Running an Agent Relay

After the server has successfully started:

1. Navigate to the *agent\_relay\_installation\bin* directory
2. Run the *run\_agentrelay.cmd* batch file (Windows), or *start\_agentrelay.cmd* (Unix/Linux).

Start the agent relay before starting any agents that will communicate through it.

## Accessing IBM uDeploy

1. Open a web browser and navigate to the host name you configured during installation.
2. Log onto the server by using the default credentials.

**User name:** *admin*

**Password:** *admin*

You can change these later by using the **Settings** tab on the IBM uDeploy web application, see ???

3. Activate the license. A license is required in order for the agents to connect to the server. Without a license, IBM uDeploy will be unable to run deployments. For information about acquiring and activating a license, see ???.



---

# Quick Start—helloWorld Deployment

This section gets you started by providing immediate hands-on experience using key product features. The *helloWorld* walk-through demonstrates how to create a simple deployment using out-of-the-box features.

## Note

This section assumes you have installed the IBM uDeploy server and at least one agent. For the walk-through, the agent can be installed on the same machine where the server is installed. If the agent or server have not been installed, see *Installing and Upgrading Servers and Agents* for installation information.

In outline, deployments are done by performing the following steps:

- **Define Components**

Components represent deployable artifacts: files, images, databases, configuration materials, or anything else associated with a software project. Components have versions which ensure that proper component instances are deployed. See *Components* for more information about creating components.

- **Define Component Processes**

Component processes operate on components, usually by deploying them. Each component must have at least one component process defined for it. For *helloWorld* you will create a single component that contains a number of text-type files (artifacts), and define a simple process that moves—deploys—the artifacts.

- **Define Application**

An application brings together all deployment components by identifying its components and defining a process to move them through a target environment (by running component processes, for instance). See *Applications* for more information about creating applications.

- **Configure Environment**

An environment is a collection of resources that represent deployment targets--physical machines, virtual machines, databases, J2EE containers, and so on. Each application must have at least one environment defined for it.

- **Identify Agent**

Agents are distributed processes that communicate with the IBM uDeploy server and perform the actual work of deploying artifacts. Generally, an agent is installed on the host where the resources it manages reside. Agents are associated with applications at the environment level.

## Creating Components

Components contain the artifacts--files, images, databases, etc.--that you manage and deploy. When creating a component, you:

1. Identify source type.

First, you define the artifacts' source type (all artifacts must be of the same type) and identify where they are stored.

2. Import a version.

After you identify the artifacts, they are imported into the artifact repository, CodeStation. Artifacts can be imported manually or automatically. When artifacts are imported, they are assigned a version ID, which enables multiple versions to be kept and managed. Snapshots, for example, can employ specific versions.

3. Define process.

The process defines how the component artifacts are deployed. A process is designed by assembling plug-in steps. A plug-in step is a basic unit of automation. Steps replace most deployment scripts and/or manual processes. Processes are designed using the drag-and-drop process editor.

## ***helloWorld* Deployment**

The *helloWorld* deployment moves some files on the local file system to another location on the file system, presumably a location used by an application server. *helloWorld* is a very simple deployment but it uses many key product features—features you will use every day.

Plug-ins provide integration with many common deployment tools and application servers. In addition to Start and Finish steps, each process has at least one step, which can be thought of as a distinct piece of user-configurable automation. By combining steps, complex processes can be easily created. Plug-ins are available for Subversion, Maven, Tomcat, WebSphere, and many other tools.

## **A Note Before You Begin**

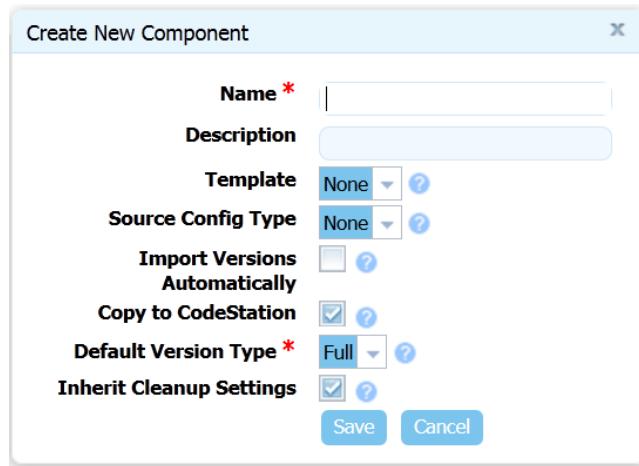
You can read the walk-through without actually performing the steps, or you can perform them as you read along. If you want to perform the steps as we go, do the following before starting:

1. Create a directory somewhere on your system named *helloWorld*.
2. Within *helloWorld* create a sub-directory named anything you like. I named mine *hello*.
3. Within the subdirectory place several—say 5—files. For speed, text-type files should be used. These files represent the artifacts that will be deployed. We will create a component that contains these files.

Reminder: If you want to perform the exercise steps, ensure that you have an active agent installed.

## ***helloWorld* Component Version**

1. Display the Create New Component dialog (Home > Components > Create New Component).

**Figure 1. Create New Component Dialog**A screenshot of the 'Create New Component' dialog box. It has a title bar with a close button. The dialog contains several fields and checkboxes. 'Name \*' is a text field. 'Description' is a text area. 'Template' is a dropdown menu with 'None' selected. 'Source Config Type' is a dropdown menu with 'None' selected. 'Import Versions Automatically' is a checkbox. 'Copy to CodeStation' is a checked checkbox. 'Default Version Type \*' is a dropdown menu with 'Full' selected. 'Inherit Cleanup Settings' is a checked checkbox. There are 'Save' and 'Cancel' buttons at the bottom right.

The initially displayed fields are the same for every source type. Other fields appearing depend on the source type and are displayed after a value is selected in the *Source Config Type* field.

2. Enter *helloWorld* in the Name field.

The name is used when assembling the application. If the component will be used by more than one application, the name should be generic rather than project-specific. For components that are project-specific, a name that conveys something meaningful about the project should be used.

3. Enter a description in the Description field.

The optional description can be used to convey additional information about the component. If the component is used by more than one application, for example, entering "Used in applications A and B" can help identify how the component is used. If you are unsure about what to enter, leave the field blank. You can always return to the component and edit the description at any time. In an attempt to appear hip, I entered *Euro store* for my component.

For this exercise, ignore the Template field. Templates provide a way to reuse commonly used component configurations. For information about templates, see the section called "Component Templates".

4. Select *File System (Versioned)* from the Source Config Type field.

Selecting a value displays several fields required by the selected type.

**Figure 2. Source Config Type**

The screenshot shows the 'Create New Component' dialog box. The fields are as follows:

- Name \***: helloWorld
- Description**: (empty)
- Template**: None
- Source Config Type**: File System (Versioned)
- Base Path \***: c:\helloWorld
- Preserve Execute Permissions**: ☐
- Import Versions Automatically**: ☒
- Copy to CodeStation**: ☒
- Default Version Type \***: Full
- Inherit Cleanup Settings**: ☒

Buttons: Save, Cancel

*File System (Versioned)* is one of the simplest configuration options and can be used to quickly set-up a component for evaluation purposes, as we do here.

5. Complete this option by entering the path to the artifacts.

In our example, the artifacts are stored inside the subdirectory created earlier. *File System (Versioned)* assumes that subdirectories within the base directory are distinct component versions, which is why we placed the files (artifacts) inside a subdirectory. *File System (Versioned)* can automatically import versions into CodeStation. If a new subdirectory is discovered when the base directory is checked, its content is imported and a new version is created.

6. Check the Import Versions Automatically check box.

When this option is selected, the source location will be periodically polled for new versions. If this option is not selected, you will have to manually import a new version every time one becomes available. The polling interval is controlled by the Automatic Version Import Check Period (seconds) field on the Settings pane (*Home > Settings > System*). The default value is 15 seconds.

7. Ensure the Copy to CodeStation check box is selected.

This option, which is recommended and selected by default, creates a tamper-proof copy of the component's artifacts and stores them in the embedded artifact management system, CodeStation. If you already have an artifact repository, such as AnthillPro or Maven, you might leave the box unchecked.

For this exercise, you can accept the default values for the remaining options and save your work.

8. After the interval specified by the system settings, the initial component version (the files inside the subdirectory created earlier) should be imported into CodeStation. To ensure the artifacts were imported, display the Versions pane (*Home > Components > helloWorld > Versions > version\_name*). This pane displays all versions for the selected component. If things went well, the

artifacts will be listed. The base path, as you will recall, is `C:\helloWorld`. Within *helloWorld* is the single subdirectory (*hello* on my machine).

**Figure 3. Imported Artifacts**

Home > Components > helloWorld > Versions > Version: hello

### Version: hello

Main Edit Properties

#### Statuses

Status	Description	Created	By	Actions
No statuses have been assigned to this version. - <a href="#">Refresh</a>				

[Add a Status](#)

#### Artifacts

Total: 12 KB (5 files)

Name	Size	Last Modified	Version	Actions
settings_license.xml	2.3 KB	1/30/12 12:38 PM	1	<a href="#">Download</a>
settings_network.xml	2.3 KB	1/30/12 12:39 PM	1	<a href="#">Download</a>
settings_notifications.xml	2.3 KB	1/30/12 12:40 PM	1	<a href="#">Download</a>
settings_properties.xml	2.3 KB	1/30/12 12:41 PM	1	<a href="#">Download</a>
settings_system.xml	2.3 KB	1/30/12 12:42 PM	1	<a href="#">Download</a>

## Component Process

Once a component has been created and a version imported, a process to deploy the artifacts—called a component process—is defined.

### To Configure the helloWorld Component Process:

1. Display the Create New Process dialog for the *helloWorld* component (Home > Components > helloWorld > Processes > Create New Process).

**Figure 4. Create New Process**

Create New Process

**Name \*** hello\_worldInstall

**Description**

**Process Type \*** Deployment ?

**Inventory Status \*** Active ?

**Default Working Directory \*** \${p:resource/work.dir}/\${p:com

**Required Component Role** None ?

Save Cancel

2. In the Create New Process dialog, enter a name in the Name field.

The name and description typically reflect the component's content and process type.

3. Enter a meaningful description in the description field.

If the process will be used by several applications, you can specify that here.

4. Accept the default values for the other fields.

You might be wondering what the Default Working Directory field does. This field points to the working directory (for temporary files, etc.) used by the agent running the process. The default value resolves to *agent\_directory\work\component\_name\_directory*. The default properties work for most components; you might need to change it if a component process cannot be run at the agent's location.

When you are done, save your work.

So far you have created a place-holder for the actual process you will define next. The name you gave the process is listed on the component's Process pane. A component can have any number of processes defined for it.

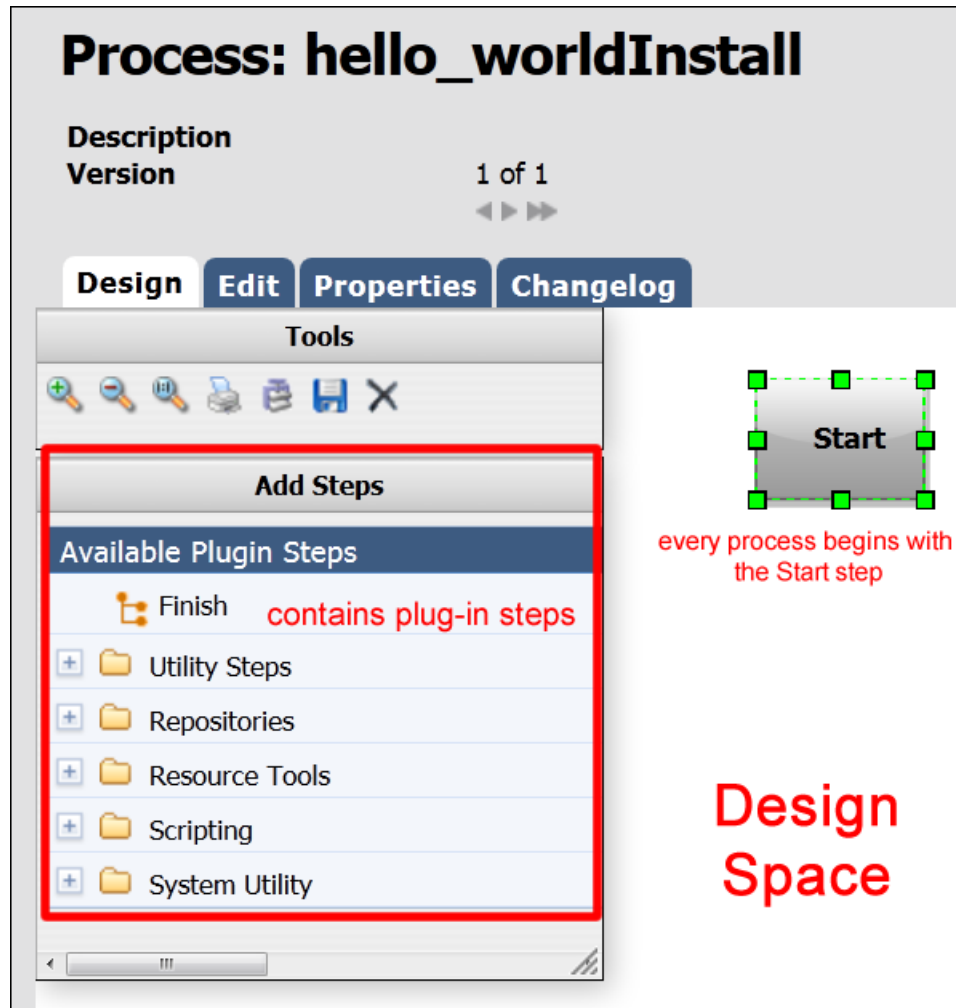
## ***helloWorld* Process Design**

Now we will complete the process by defining the actual plug-in steps. In addition to the Start and Finish steps which are part of every process, a process must have at least one additional step. The steps are defined with the Process Design pane. You define the steps by dropping them onto the design area and arranging them in the order they are to be executed.

### **To Define the *helloWorld* Process Steps:**

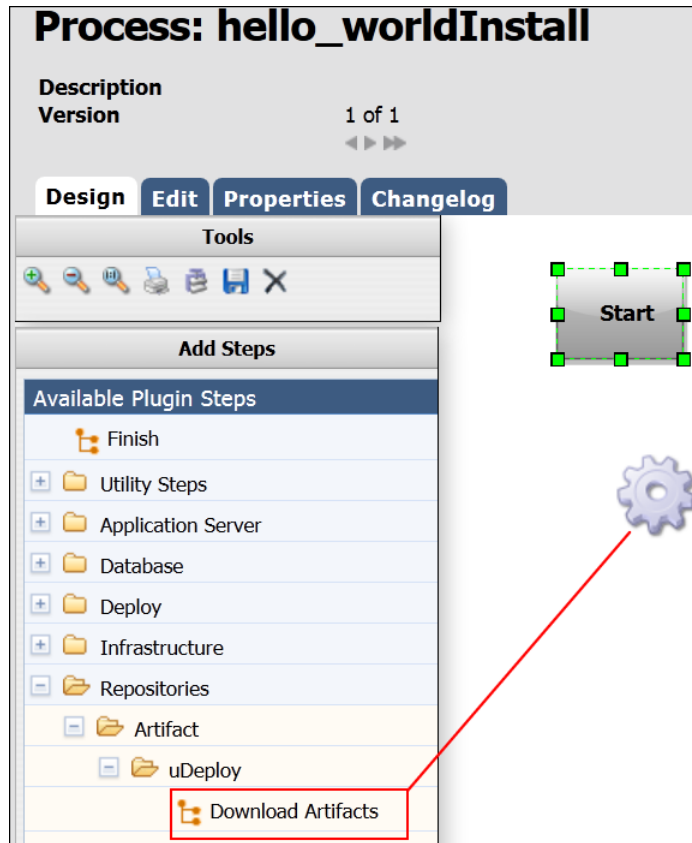
1. Display the Process Design pane for the process created earlier (*Home > Components > helloWorld > process\_name*).

Figure 5. Process Design Pane



The steps are listed in the Available Plug-in Steps list-box. Take a moment to expand the listings and review the available steps. Many commonly used plug-in steps are available out-of-the-box.

2. In the Available Plug-in Steps box, expand the *IBM uDeploy* menu item (*Repositories > Artifact > IBM uDeploy*).
3. Drag the *Download Artifacts* step onto the design space and release it. For now, don't worry about where the step is released—a step's position in the workflow is defined after its parameters are configured.

**Figure 6. Adding a Step to the Process**

The Edit Properties dialog is displayed when the mouse-pointer is released over the design space.



**Figure 7. Edit Properties Dialog**

The 'Edit Properties' dialog box is shown with the following fields and values:

- Name \***: Download Artifacts
- Directory Offset \***: .
- Includes \***: \*\*/\*
- Excludes**: (empty)
- Sync Mode**: Sync
- Full Verification**: ☒
- Set File Execute Bits**: ☐
- Verify File Integrity**: ☐
- Allow Failure**: ☐
- Working Directory**: (empty)
- Post Processing Script**: Step Default
- Precondition**: (empty)
- Use Impersonation**: ☐

Buttons at the bottom: Save, Cancel.

This dialog displays all configurable parameters associated with the selected step.

For this exercise, we can achieve our goal by entering data into a single field—Directory Offset. Recall that the goal for this ambitious deployment is to move the source files in the base directory to another location. As you might guess, several methods for accomplishing this are available. Pointing the Directory Offset field to the target location is one of the simplest.

4. In the Directory Offset field, enter the path to the target directory. Because IBM uDeploy can create a directory during processing, you specify any target directory. I entered `c:\hello` which did not exist on my system, and let IBM uDeploy create it for me.

If the field is left blank, the process will use the working directory defined earlier. Entering the path overrides the previous value and will cause the source files to be moved—deployed—to the entered location when the process runs. The default value would move (download) the files to `agent_directory\work\component_name_directory`.

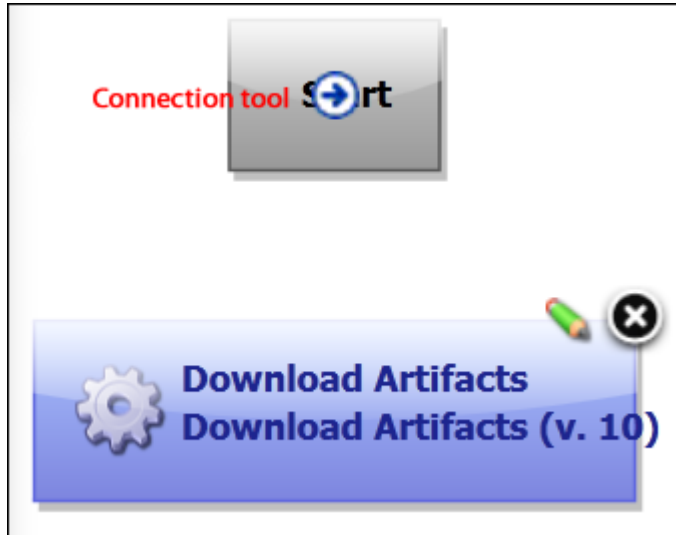
After entering the target path, save your work and close the dialog box.

5. Next, the step must be positioned within the process workflow. There's no requirement that a step be positioned immaculately after it's created; you could place several more before defining their positions, but because this is the only step we are adding, it makes sense to define its position now.

A step's position in the workflow is defined by dragging connection arrows to/from it. The arrows define the direction of the workflow.

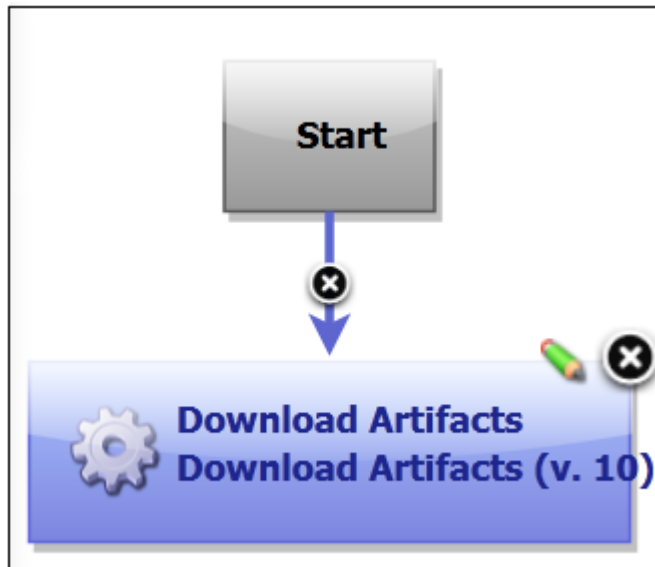
Hover the mouse pointer over the *Start* step to display the connection tool, as shown in the following illustration. Each step has a connection tool which is used to connect it to other steps.

**Figure 8. Connection Tool**

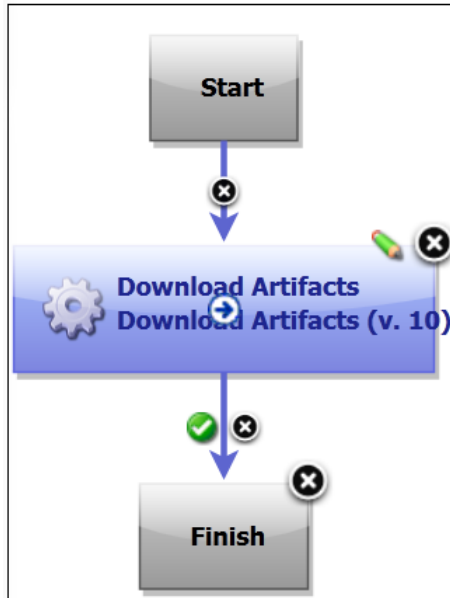


Grab the connection tool and drag it over the *Download Artifacts* step then release it. A connection arrow connects the two steps. The arrow indicates the direction of process flow—from the originating step to the destination step.

**Figure 9. Finished Connection**



6. Complete the process by connecting the *Download Artifacts* step to the *Finish* step. A step can have more than one arrow originating from it and more than one connecting to it.

**Figure 10. Completed Process**

7. Save the component by using the Save tool on the Tools menu.

Once the process steps are defined, the final task is to define an application that uses the component—and the component process you just created.

## ***helloWorld* Application**

To deploy the *helloWorld* component, you must create an application. An application, as used by IBM uDeploy, is a mechanism that deploys components into environments using *application* processes—processes similar to the component process just defined.

To create an application, you: identify the components it controls (an application can manage any number of components); define at least one environment into which the components will be deployed; and create a process to perform the work. An environment maps components to agents and handles inventory, among other things.

An application process is similar to but not identical with a component process. While application processes consists of steps configured with the process editor, like component processes, they are primarily intended to direct underlying component processes and orchestrate multi-component deployments. The Install Component step, which we will use shortly, enables you to select a component process from among those defined for each component (remember that a component can have more than one process defined for it).

You perform a deployment by running an application process for a specific environment.

You might be wondering why you need to create an application-level process when the process you created for the component should be able to perform the deployment by itself. While individual component processes can be run outside an application process, an environment must still be defined (environments are defined at the application level) and an agent associated with it. For a single-component deployment like *helloWorld*, an application-level process might not be required. You might also want to skip an

application-level process when you are testing or patching a component. But for non-trivial deployments, and especially for deployments that have more than one component, you will want to create one or more application-level processes.

## Creating an Application

**To create an application:**

1. Display the Create New Application dialog (Home > Applications > Create New Application [button]). Unlike the Create New Component dialog box where some fields vary depending on the artifacts' source, none of the fields here are variable.
2. Enter a name and description.

I entered *helloWorld\_application*. While there is no naming requirements, the number of associated items—components, processes, applications, environments, etc.—can become large, so it's useful to employ a scheme that makes it easy to identify related items.

3. Accept the default value of *None* from the Notification Scheme drop-down list box, and save the application.

IBM uDeploy integrates with LDAP and e-mail servers which enables it to send event-based notifications. For example, the default notification scheme will send an e-mail (if an email server has been configured, see ???) when a deployment finishes. Notifications can also play a role in deployment approvals. See ??? for information about security roles.

## Adding the helloWorld Component to the Application

After the application is saved, we identify the component—helloWorld—it will manage. While we have only one component to deploy, an application can manage any number of components.

1. Display the Add a Component dialog for the application just created, *helloWorld\_application* in my case (Home > Applications > *helloWorld\_application* > Components > Add Component [button]).
2. Select *helloWorld* from the Select a Component drop-down list box, then save your selection.

The simple act of selecting a component accomplishes a lot. The the component processes defined for the component become available to the application, for example, and many application process steps will have default values set to values defined by the *helloWorld* component.

## Adding an Environment to the Application

Before an application can run, it must have at least one environment created for it. An environment defines the agents used by the application, among other things.

1. Display the Create New Environment dialog (Home > Applications > *helloWorld\_application* > Create New Environment).

**Figure 11. Create New Environment**

The screenshot shows a 'Create New Environment' dialog box with the following elements:

- Name \***: A text input field.
- Description**: A text input field.
- Require Approvals**: A checkbox with a help icon.
- Lock Snapshots**: A checkbox with a help icon.
- Color**: A grid of 48 color swatches arranged in 6 rows and 8 columns.
- Inherit Cleanup Settings**: A checked checkbox with a help icon.
- Buttons**: 'Save' and 'Cancel' buttons at the bottom right.

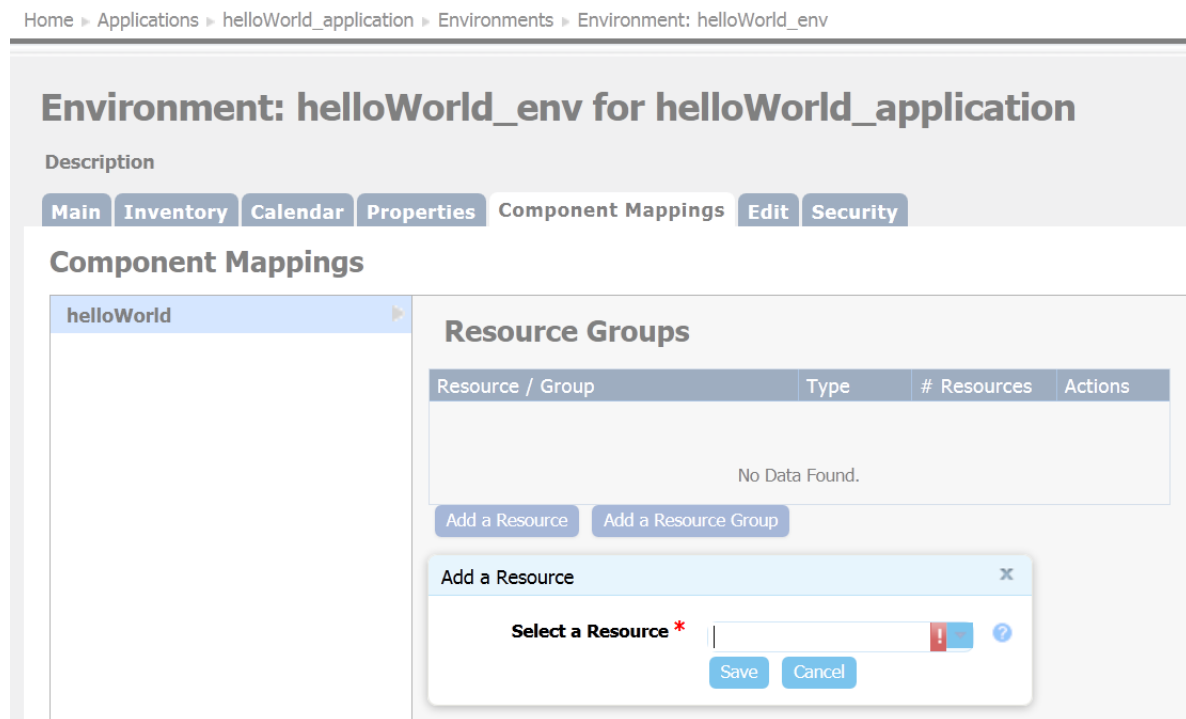
2. Use the Create New Environment dialog to define the environment:

- The value in the Name field will be used in the deployment.
- If you check the Require Approvals check box, approvals will be enforced. See *Deployments* for information about the approval process. This is our first deployment so an uncontrolled environment will do fine—leave the box unchecked.
- Leave the Lock Snapshots check box unchecked. When this is checked, snapshots added to this environment will be locked (for this environment). A snapshot is a grouping of component versions and processes, see the section called “Snapshots”.
- Selecting a color provides a visual identifier for the environment in the UI. Typically, each environment will be assigned its own color.
- Leave the Inherit Cleanup Settings check box checked. Clean-up refers to archiving component versions. When a component is archived, its artifacts are combined into a ZIP file and saved. The corresponding component is removed from CodeStation. When checked, settings are inherited from the system settings, otherwise they are inherited from the application's components, see ???.

3. Next, add an agent that will execute the application's process steps. Display the Add a Resource dialog (Applications > helloWorld\_application > Environments > Environment: name > Component Mappings > Add a Resource).

Select any of the agents that were created when IBM uDeploy was installed.

While our example uses only a single resource, deployments can use many resources and *resource groups*. Resource groups provide a way to combine resources, which can be useful when multiple deployments use overlapping resources. See *Resources* for information about resource groups.

**Figure 12. Component Mappings**

## Adding a Process to the Application

Now that our application has an environment, we can create an application-level process that will perform the deployment.

1. Display the Create an Application Process dialog (Applications > helloWorld\_application > Processes > Create New Process).
2. Enter a name in the Name field.

Accept the default values for the other fields:

- The Required Application Role drop-down field is used to restrict who can run this process. The default value, None, means anyone can run the process. The available options are derived from the IBM uDeploy Security System. For information about security roles, see ???
- The Inventory Management field determines how inventory for the application's components are handled. If you want to manually control inventory, you would select the *Advanced* option. See ??? for information about inventory management.

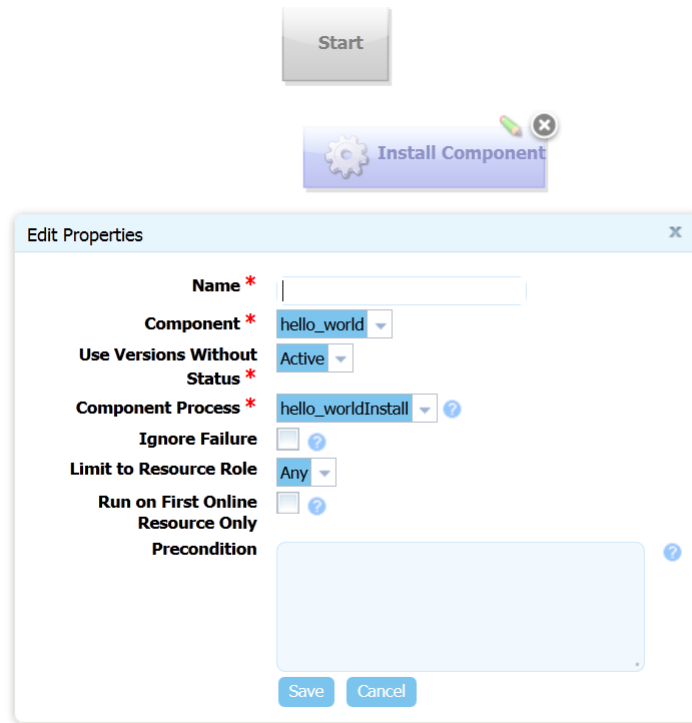
3. Save your work when you are finished.

## Designing the Process Steps

To create an application-level process, you define the individual steps as you did earlier (see the section called “Component Processes”) when you used the Process Design pane to create the *helloWorld* component process.

1. Display the Process Design pane (Applications > application\_name > Processes > process\_name). The out-of-box process steps are listed in the Add a Component Process list box.
2. Drag the *Install Component* step onto the design area and release. The Edit Properties dialog is displayed.

**Figure 13. Edit Properties Dialog**



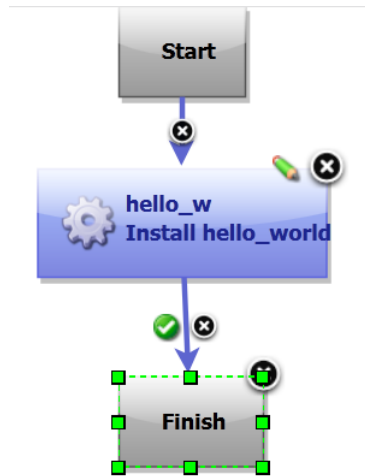
Select a component from the **Component** drop-down list box. If you followed the *Quick Start Guide*, the *helloWorld* component will be listed.

If we wanted this application to install multiple components, we could add a separate Install Component step for each one.

3. Use the Component Process list box to select the component process you created earlier. All processes defined for the selected component are listed. If the component had another process that deployed it to a different location, you could add another Install Component step that used that process—simultaneously installing the component into two different locations.

Accept the default values for the other fields (see *Applications* for information about the other fields), and click **Save**.

4. Connect the step to the *Start* and *Finish* steps.

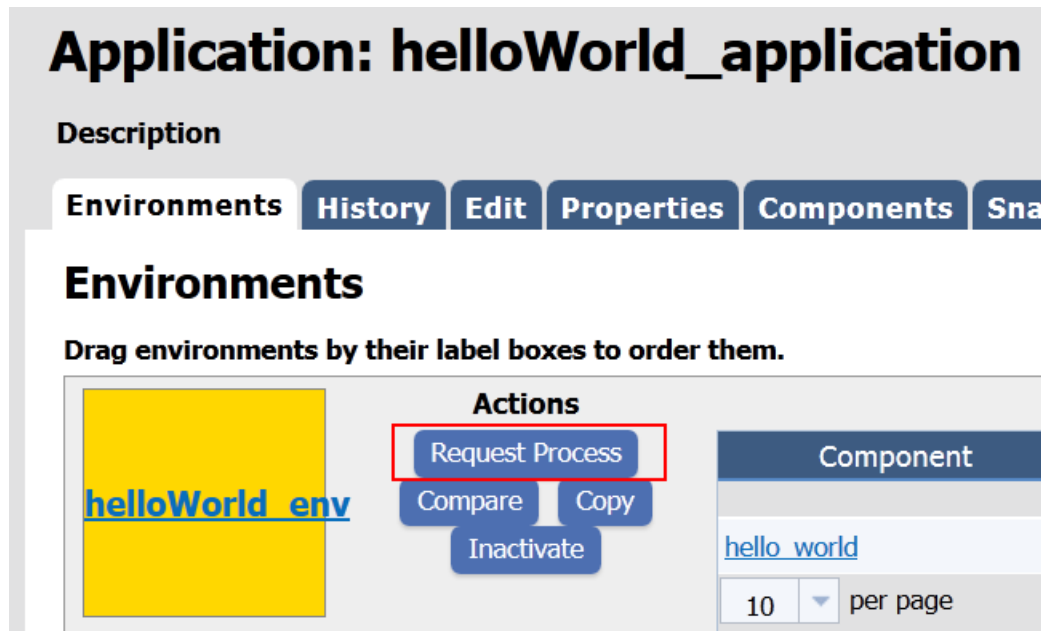
**Figure 14. Finished Application Process**

5. Save the process by clicking the Save tool on the Tools bar.

## Running the Application

Now that the component, environment, and process are complete, you are ready—finally!—to run the application.

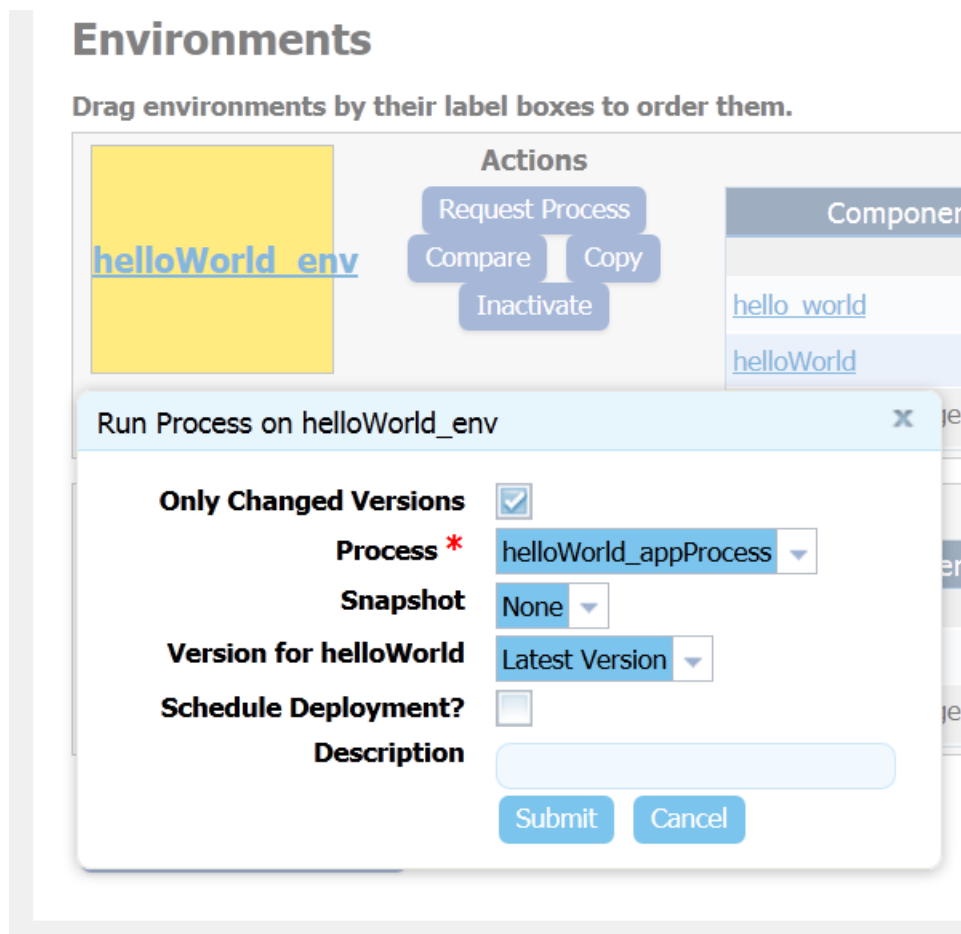
1. On the Application pane, click the Request Process button for the environment you created earlier.

**Figure 15. Request Process**

2. On the Run Process dialog:



- Select the process you created from the Process drop-down list box. Applications can have more than one process defined for them..
- Select Latest Version from the Version drop-down list box. This option ensures that the latest (or first and only) version is affected.

**Figure 16. Run Process Dialog**

3. Click Submit to run the application.

The Application Process pane is displayed. This pane displays the application's status.

**Figure 17. Application Process Request**

Log <b>Properties</b> Manifest					
<a href="#">Expand All</a> <a href="#">Collapse All</a>		Sort By: <a href="#">Graph Order</a> <a href="#">Start Time</a>			
Step	Progress	Start	Duration	Status	Actions
<b>helloAppPro</b>	<b>1 of 1</b>	3:42:47 PM	0:00:03	Success	
<code>tph-agent</code>	<b>1 of 1</b>	3:42:47 PM	0:00:02	Success	
<code>hello</code>		3:42:47 PM	0:00:02	Success	<a href="#">Details</a>
<b>Total Execution</b>	<b>1 of 1</b>	<b>3:42:47 PM</b>	<b>0:00:03</b>	<b>Success</b>	

Take a few moments to examine the information on this pane; hopefully, you will see a Success message. To see additional information (Output Log, Error Log, Application Properties), click the Details link.

---

# Using IBM uDeploy

---

---

# Components

Components represent deployable items along with user-defined processes that operate on them, usually by deploying them. Deployable items, or artifacts, can be files, images, databases, configuration materials, or anything else associated with a software project. Artifacts can come from a number of sources: file systems, build servers such as AnthillPro, as well as many others. When you create a component, you identify the source and define how the artifacts will be brought into IBM uDeploy.

## Component Versions and the CodeStation Repository

After defining a component's source and processes, you import its artifacts into IBM uDeploy's artifact repository CodeStation. Artifacts can be imported automatically or manually. By default, a complete copy of an artifact's content is imported into CodeStation (the original artifacts are untouched). Each time a component is imported, including the first time, it is versioned. Versions can be assigned automatically by IBM uDeploy, applied manually, or come from a build server. Every time a component's artifacts are modified and reimported, a new version of the component is created.

## Component Processes

A component process is a series of user-defined steps that operate on a component's artifacts. Each component has at least one process defined for it and can have several. A component process can be as simple as a single step or contain numerous relationships, branches, and process switches. Component processes are created with IBM uDeploy's process editor. The process editor is a visual drag-and-drop editor that enables you to drag process steps onto the design space and configure them as you go. As additional steps are placed, you visually define their relationships with one another. Process steps are selected from a menu of standardized steps. IBM uDeploy provides steps for several utility processes, such as inventory management, and workflow control. Additional process steps are provided by plug-ins. A component process can have steps from more than one plug-in. See *Plug-ins*.

Additionally, you can create processes and configure properties and save them as templates to create new components. See the section called “Component Templates”.

# Creating Components

In general, component creation is the same for all components. When creating a component, you:

1. Define source type.

You name the component and identify the artifacts' source, such as AnthillPro, a file system, or Subversion. A component can contain any number of artifacts but they must all share the same source.

2. Assemble process(es).

A process defines what IBM uDeploy does with the component's artifacts. A process might consist of any number of steps, such as starting and starting servers, and moving files. In addition to deploying, other processes can import artifacts and perform various utility tasks.

To reiterate, then, a component consists of artifacts all sharing the same source type, plus one or more processes. In addition to hand-crafting a component, you can use a template to create one (see the section called “Component Templates”), or you can import a component directly (see the section called “Importing/Exporting Components”).

**To create a component:**

1. Display the Create New Components dialog (Home > Components > Create New Component). Several fields are the same for every source, while others depend on the source type selected with the Source Config Type field.

**Figure 18. Create New Component Dialog**

2. Define standard parameters. The fields in the following table are available for every source type. If you select a value in the Source Config Type field, fields specific to the selected type are also displayed.

**Table 8. Fields Available for All Source Types**

Field	Description
<b>Name</b>	Identifies the component; appears in many UI features. Required.
<b>Description</b>	The optional description can be used to convey additional information about the component. If the component is used by more than one application, for example, entering "Used in applications A and B" can help identify how the component is used.
<b>Template</b>	<p>A component template enables you to reuse component definitions; components based on templates inherit the template's source configuration, properties, and process. Any previously created templates are listed. A component can have a single template associated with it. The default value is <i>None</i>.</p> <p>If you select a template, the Template Version field is displayed which is used to select a template version. By controlling the version, you can roll-out template changes as required. The default value is Latest Version which means the component will automatically use the newest version (by creation date). See the section called "Component Templates".</p> <p><b>Note</b></p> <p>If you select a template that has a source configured for it, the dialog box will change to reflect values defined</p>

Field	Description
	for the template. Several fields, including the Source Config Type field, will become populated and locked.
<b>Source Config Type</b>	Defines the source type for the component's artifacts; all artifacts must have the same source type. Selecting a value displays additional fields associated with the selection. Source-dependent fields (see <i>Component Source Configuration</i> ) are used to identify and configure the component's artifacts. If you selected a template, this field is locked and its value is inherited from the template.
<b>Import Automatically</b> <b>Versions</b>	If checked, the source location is periodically polled for new versions; any found are automatically imported. The default polling period is 15 seconds, which can be changed with the System Settings pane. If left unchecked, you can manually create versions by using the Versions pane. By default, the box is unchecked.
<b>Copy to CodeStation</b>	This option—selected by default—creates a tamper-proof copy of the artifacts and stores them in the embedded artifact management system, CodeStation. If unchecked, only meta data about the artifacts are imported. UrbanCode, an IBM Company recommends that the box be checked.
<b>Default Version Type</b>	Defines how versions are imported into CodeStation. Full means the version is comprehensive and contains all artifacts; Incremental means the version contains a subset of the component's artifacts. Default value is: Full. Required.
<b>Inherit Cleanup Settings</b>	Determines how many component versions are kept in CodeStation, and how long they are kept. If checked, the component will use the values specified on the System Settings pane. If unchecked, the Days to Keep Versions (initially set to -1, keep indefinitely) and Number of Versions to Keep (initially set to -1, keep all) fields are displayed, which enable you to define custom values. The default value is checked.

3. If you select a source type, enter values into the source-specific field. See *Component Source Configuration* for information about the source types and parameters.
4. When finished, save your work. Saved components are listed in the Component pane.

## Importing/Exporting Components

Components can be imported and exported. Importing/exporting can be especially useful if you have multiple IBM uDeploy servers, for example, and need to quickly move or update components.

### Exporting Components

Exporting a component creates a JSON file (file extension `json`) that contains the component's source configuration information, properties, and processes. For information about JSON, see <http://www.json.org/>.

#### To export a component:

On the Components pane (Home > Components), click the Export link in the Actions field. You can load the file into a text editor, or save it. If you save it, a file is created with the same name as the selected component, for example, `helloWorld.json`.

## Importing Components

When you import a component, you can create an entirely new component or upgrade an existing one. Additionally, if the imported component was created from a template, you can use it or create a new one.

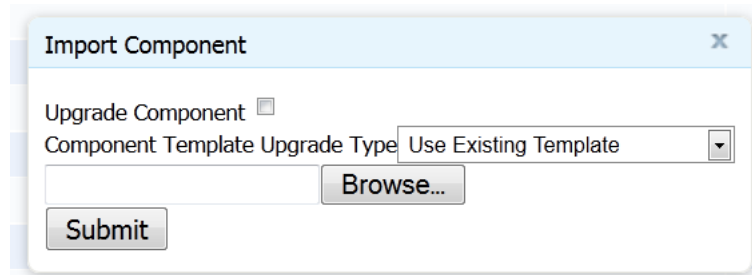
### Note

If the imported component has the Import Versions Automatically parameter set to true, the new component will automatically import component versions as long as the artifacts are accessible to the importing server.

### To Import a Component

1. Display the Import Component dialog (Components > Import Component [button]).

**Figure 19. Import Component Dialog**



2. Enter the path to the JSON file containing the component definition or use the Browse button to select one.
3. If you want to upgrade an existing component, check the Upgrade Component check box. To create a new component, leave the box unchecked.

If the component's name in the JSON file (not the name of the file itself) matches an existing component, the component's parameters are updated with the new values, and new items—such as processes—are added. If the name of the component is not found, the command has no effect.

### Note

The component's name is the first parameter in the JSON file; for example,

```
"name": "helloWorld",
```

.

4. If the imported component was originally created from a template, use the Component Template Upgrade Type drop-down box to specify how you want to use the template. For these options, the template must be on the importing server. If the imported component was not created from a template, these options are ignored.
  - To use the imported component's template, select Use Existing Template. The new component will be an exact copy of the imported one and contain a pointer to the imported component's template. This option is especially useful if you are importing a lot of components based on the same template.

If you are upgrading, the component will also point to the imported template.

- To create a new template, select Create New Template. The new component will be an exact copy of the imported one and contain a pointer to the newly created template (which is based on the imported component's template).

If you are upgrading a component, a new template is also created used.

- When you want to create a fresh installation and ensure a template is *not* on the importing server, select Fail if Template Exists. If you are creating a component, it will create both a new component and template unless the template already exists, in which case the component is not imported.

If you are upgrading a component, the upgrade will fail if the imported component's template already exists.

- To ensure the template is on the importing server, select Fail if Template Does Not Exist. If you are creating a component, it will create both a new component and template unless the template does not exist, in which case the component is not imported.

If you are upgrading a component, the upgrade will fail if the imported component's template does not exist on the importing server.

- To upgrade the template, select Upgrade if Exists. This option creates a new component and upgrades the template on the importing server. If the template does not exist, a new one is created.

5. Click Submit.

## Component Properties

There are three types of component properties available: custom, environment, and version (another type, component, is defined by template and becomes part of any component created from the template, see the section called “Component Template Properties”). Property versions (changes) are maintained and remain available.

The three types can be defined on the component's Properties pane (*Components* > [selected component] > *Properties*). The three types are described in the following table.

**Table 9. Component Properties**

Type	Description
Properties	Custom property; can be used in scripts and plug-ins. Those inherited from templates cannot be modified on the component level.  Referenced: <code>\${p:component/propertyName}</code> .
Environment	Available to environments that use the component. The property will appear on the environment's Component Mappings pane ( <i>Applications</i> > [selected application] > <i>Environments</i> > [selected environment] > <i>Component Mappings</i> ), see the section called “Application Environments”. Each property must have a type: <ul style="list-style-type: none"> <li>• <i>Text</i></li> </ul>



Type	Description
	<p>Enables users to enter text characters.</p> <ul style="list-style-type: none"> <li>• <i>Text Area</i></li> </ul> <p>Enables users to enter an arbitrary amount of text, limited to limited to 4064 characters.</p> <ul style="list-style-type: none"> <li>• <i>Check Box</i></li> </ul> <p>Displays a check box. If checked, a value of <i>true</i> will be used; otherwise the property is not set.</p> <ul style="list-style-type: none"> <li>• <i>Select</i></li> </ul> <p>Requires a list of one or more values which will be displayed in a drop-down list box. Enables a single selection. <i>Note: not currently implemented.</i></p> <ul style="list-style-type: none"> <li>• <i>Multi Select</i></li> </ul> <p>Requires a list of one or more values which will be displayed in a drop-down list box. Enables multiple selections.</p> <ul style="list-style-type: none"> <li>• <i>Secure</i></li> </ul> <p>Used for passwords. Similar to Text except values are redacted.</p> <p>A value set on component environment overrides one with the same name set directly on an environment property. Component environment properties enable you to centralize properties, tracking type and default values, for instance. Environment properties provide ad-hoc lists of property=value pairs.</p> <p>Referenced: <math>\{p:environment/propertyName\}</math>.</p>
Version	<p>Available to every component version (<i>Components &gt; [selected component] &gt; Versions &gt; [selected version] &gt; Properties</i>). Values can be set at the individual version level. Each property must have a type (described above).</p> <p>Referenced: <math>\{p:version/propertyName\}</math>.</p>

## Component Versions

Each time a component's artifacts are imported into the repository, including the first time, it is versioned. Versions can be assigned automatically by IBM uDeploy, applied manually, or come from a build server. Every time a component's artifacts are modified and reimported, a new version of the component is created. So a component might have several versions in CodeStation and each version will be unique.

A version can be full or incremental. A full version contains all component artifacts; an incremental version only contains artifacts modified since the previous version was created.

## Importing Versions Manually

1. Display the Version pane for the component you want to use (Components > [select component] > Versions).

**Figure 20. Component Version Pane**

**Component: helloWorld**

**Used By**  
[del](#) (*Inactive*), [fileVVV](#), [helloWorld](#), [helloWorldComp](#), [helloWorldImp](#),  
[helloWorld\\_application](#)

[History](#) [Edit](#) [Inventory](#) [Calendar](#) [Properties](#) [Templates](#) **Versions** [Processes](#) [Tasks](#) [Changes](#) [Security](#)

**Versions**

Version	Latest Status	Type	Created By	Date	Description	Actions
<a href="#">Show Filters</a>						
<a href="#">21</a>		Full	admin	8/6/12 11:38 AM		<a href="#">Inactivate</a> <a href="#">Delete</a>
<a href="#">hello</a>		Full	admin	7/10/12 4:12 PM		<a href="#">Inactivate</a> <a href="#">Delete</a>
<a href="#">installation</a>		Full	admin	7/10/12 4:10 PM		<a href="#">Inactivate</a> <a href="#">Delete</a>
<a href="#">images</a>		Full	admin	7/10/12 4:10 PM		<a href="#">Inactivate</a> <a href="#">Delete</a>
<a href="#">helloWorld</a>		Full	admin	7/10/12 4:10 PM		<a href="#">Inactivate</a> <a href="#">Delete</a>
<a href="#">Doc1_files</a>		Full	admin	7/10/12 4:10 PM		<a href="#">Inactivate</a> <a href="#">Delete</a>
<a href="#">aa</a>		Full	admin	7/10/12 4:10 PM		<a href="#">Inactivate</a> <a href="#">Delete</a>
<a href="#">3</a>		Full	admin	7/10/12 4:10 PM		<a href="#">Inactivate</a> <a href="#">Delete</a>

10 per page 8 records - [Refresh](#) [Print](#)

☐ Show Inactive Versions

[Import New Versions](#)

All versions; statuses come from; active/inactive *Source Config Type* field.

2. Enter *helloWorld* in the Name field.

Display the Import .

3. Enter a description in the Description field.

The optional description can be used to convey additional information about the component. If the component is used by more than one application, for example, entering "Used in applications A and B" can help identify how the component is used. If you are unsure about what to enter, leave the field blank. You can always return to the component and edit the description at any time. In an attempt to appear hip, I entered *Euro store* for my component.

For this exercise, ignore the Template field. Templates provide a way to reuse commonly used component configurations. For information about templates, see the section called "Component Templates".

4. Select *File System (Versioned)* from the Source Config Type field.

Selecting a value displays several fields required by the selected type.

**Figure 21. Source Config Type**

The screenshot shows a 'Create New Component' dialog box with the following fields and values:

- Name \***: helloWorld
- Description**: (empty)
- Template**: None
- Source Config Type**: File System (Versioned)
- Base Path \***: c:\helloWorld
- Preserve Execute Permissions**: ☐
- Import Versions Automatically**: ☒
- Copy to CodeStation**: ☒
- Default Version Type \***: Full
- Inherit Cleanup Settings**: ☒

Buttons: Save, Cancel

*File System (Versioned)* is one of the simplest configuration options and can be used to quickly set-up a component for evaluation purposes, as we do here.

5. Complete this option by entering the path to the artifacts.

In our example, the artifacts are stored inside the subdirectory created earlier. *File System (Versioned)* assumes that subdirectories within the base directory are distinct component versions, which is why we placed the files (artifacts) inside a subdirectory. *File System (Versioned)* can automatically import versions into CodeStation. If a new subdirectory is discovered when the base directory is checked, its content is imported and a new version is created.

6. Check the Import Versions Automatically check box.

When this option is selected, the source location will be periodically polled for new versions. If this option is not selected, you will have to manually import a new version every time one becomes available. The polling interval is controlled by the Automatic Version Import Check Period (seconds) field on the Settings pane (*Home > Settings > System*). The default value is 15 seconds.

## Importing Versions Automatically

When this option is selected, the source location is periodically polled for new versions; any found are automatically imported. The default polling period is 15 seconds, which can be changed with the System Settings pane, see ???).

## Component Version Statuses

Component version statuses are user-managed values that can be added to component versions. Once a status is added to a version, the value can be used in component processes or application gates (see the section called “Application Gates”).

Version statuses can be applied to a component version through the user interface (*Components > [selected component] > Versions > [selected version] > Add a Status [button]*), or by the Add Status to Version plug-in step.

IBM uDeploy-provided statuses are defined in an XML file which you can freely edit to add your own values.

## Deleting Component Versions

You can delete any component version. To delete a version, use the Delete action for the version (*Components > [selected component] > Versions > [selected version] > Delete*). Deleting a version removes associated meta data from the repository; original artifacts are unaffected.

## Inactivating Component Versions

Inactive component versions remain in the repository (unlike deleted versions) but cannot be deployed. To render a component version inactive, use the Inactivate action for the version (*Components > [selected component] > Versions > [selected version] > Inactivate*).

To make an inactive version active, use the Show Inactive Versions check box and the Activate action.

## Component Processes

A component process is a series of user-defined steps that operate on a component's artifacts. Each component has at least one process defined for it and can have several. Component processes are created with IBM uDeploy's process editor. The process editor is a visual drag-and-drop editor that enables you to drag process steps onto the design space and configure them as you go. Process steps are selected from a menu of standard steps. See the section called “Process Editor”

IBM uDeploy provides steps for several utility processes such as inventory management and workflow control. Additional process steps are provided by plug-ins. Out-of-the-box, IBM uDeploy provides plug-ins for many common processes, such as downloading and uploading artifacts, and retrieving environment information. See *Plug-ins*.

A frequently used process can be saved as a template and applied to other components. See the section called “Component Templates”.

## Configuring Component Processes

A component process is created in two steps: first, you configure basic information, such as name; second, you use the process editor to assemble the process.

**To configure a component process:**

1. Display the Create New Process dialog (*Home > Components > Component: component\_name > Create New Process*).

**Figure 22. Create New Process Dialog**

2. The dialog's fields are described in the following table.

**Table 10. Create New Process Fields**

Field	Description
<b>Name</b>	Identifies the process; appears in many UI elements. Required.
<b>Description</b>	The optional description can be used to convey additional information about the process.
<b>Process Type</b>	<p>Defines the process type. Available values are:</p> <ul style="list-style-type: none"> <li>• <b>Deployment:</b> deploys a component version to the target resource and updates the inventory after a successful execution.</li> <li>• <b>Configuration Deployment:</b> configuration-only deployment with no component version or artifacts—simply applies the configuration (using properties, or configuration templates) to the target agent and updates the resource configuration inventory afterwards.</li> <li>• <b>Uninstall:</b> standard uninstall that removes a component version from a target resource and the resource's inventory.</li> <li>• <b>Operational (With Version):</b> operational process which does not add or remove any artifacts or configuration; runs arbitrary steps given a component version. Useful when you want to start or stop some service for a previously deployed component version.</li> <li>• <b>Operational (No Version Needed):</b> same as the previous type, but does not require a component version.</li> </ul> <p>Required.</p>

Field	Description
<b>Inventory Status</b>	Status applied to component versions after being successfully executed by this process. <i>Active</i> indicates the component version is deployed to its target resource; <i>Staged</i> means the component version is in a pre-deployment location. The status appears on the Inventory panes for the component itself and environments that ran the process. Required.
<b>Default Working Directory</b>	Defines the location used by the agent running the process (for temporary files, etc.). The default value resolves to <i>agent_directory\work\component_name_directory</i> . The default properties work for most components; you might need to change it if a component process cannot be run at the agent's location. Required.
<b>Required Component Role</b>	Restricts who can run the process. The available options are derived from the IBM uDeploy security system. The default value is <i>None</i> , meaning anyone can run the process. For information about security roles, see ???.

3. Save your work when you are finished. The process is listed on the Processes pane for the associated component.

## Running Component Processes

Component processes are typically run from within application processes (the section called “Application Processes”) but can also be run by generic processes (*Generic Processes*) or other component processes.

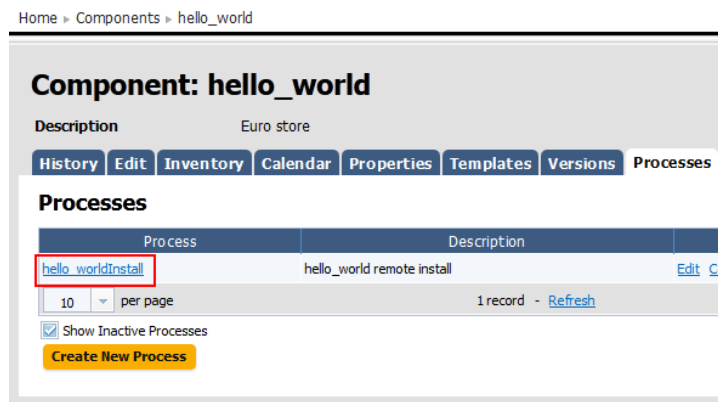
## Process Editor

After configuring a process with the **Create New Process** dialog, use the process editor to assemble the process.

## To Display the Process Editor

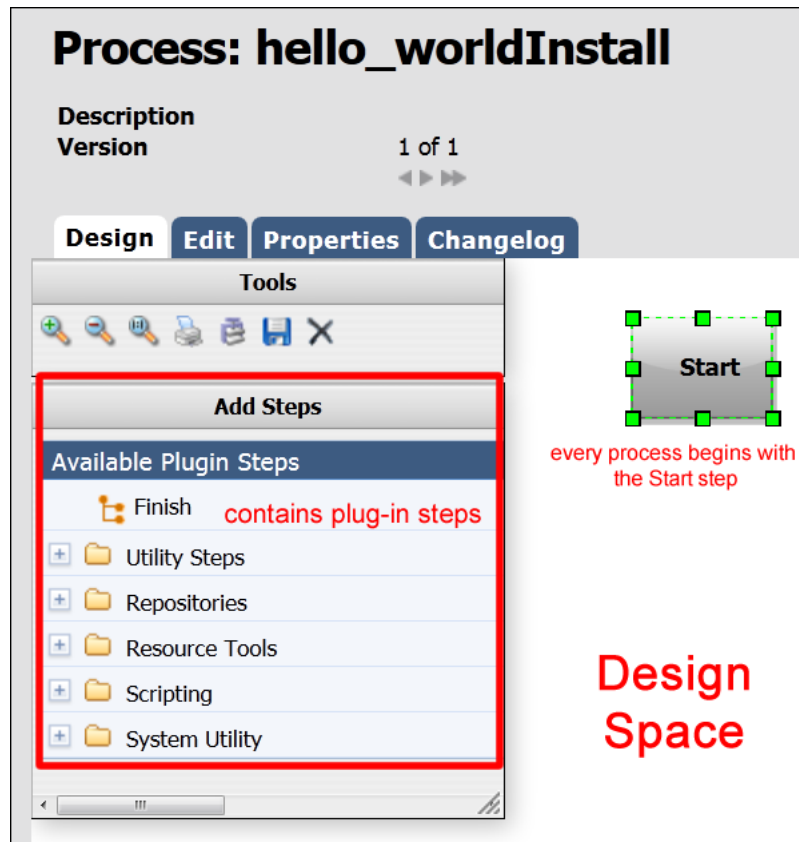
1. On the **Component: name** pane, click the **Processes** tab.
2. Click on the name of the process you want to edit.

**Figure 23. Component Processes**



The **Process Design** pane is displayed.

**Figure 24. Process Design Pane**



Available steps are listed in the **Available Plug-in Steps** list. IBM uDeploy provides several utility steps and plug-ins which are highlighted in the accompanying illustration. The illustration also shows several user-installed plug-ins.

## Using the Process Editor

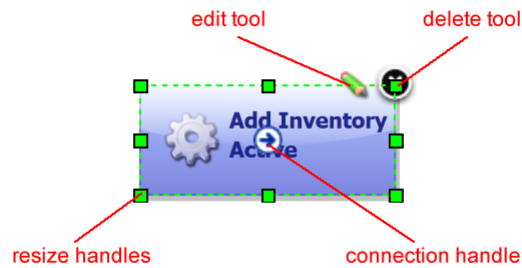
When the **Process Design** pane opens, the **Design** view is displayed. Processes are assembled with the **Design** view. Several other views can be displayed by clicking the associated tab:

**Table 11. Available Views**

View	Description
<b>Edit</b>	Displays the <b>Edit</b> view where you can change process parameters. See the section called “Component Processes”.
<b>Properties</b>	Displays the <b>Properties</b> view where you can create and change process properties.
<b>Changelog</b>	Displays the <b>Process Changelog</b> view. This view provides a record for every process change--property add or delete, and process save or delete.

In outline, processes are assembled by dragging individual steps onto the design space and configuring and connecting them as they are placed. When a step is dragged onto the design space, a pop-up is displayed that is used to configure the step. Once configured and the pop-up closed, relationships between steps are formed by dragging *connection handles* between associated steps.

**Figure 25. Typical Process Step**



Graphically, each step (except for the Start step which cannot be deleted or edited) is the same and provides:

**Table 12. Anatomy of a Step**

Item	Description
<b>edit tool</b>	displays the step configuration pop-up where you can modify configuration parameters
<b>delete tool</b>	removes the step from the design space
<b>resize handle</b>	enables you to resize the step graphic
<b>connection tool</b>	used to create connections between steps

### Note

If you delete a step, its connections (if any) are also deleted.

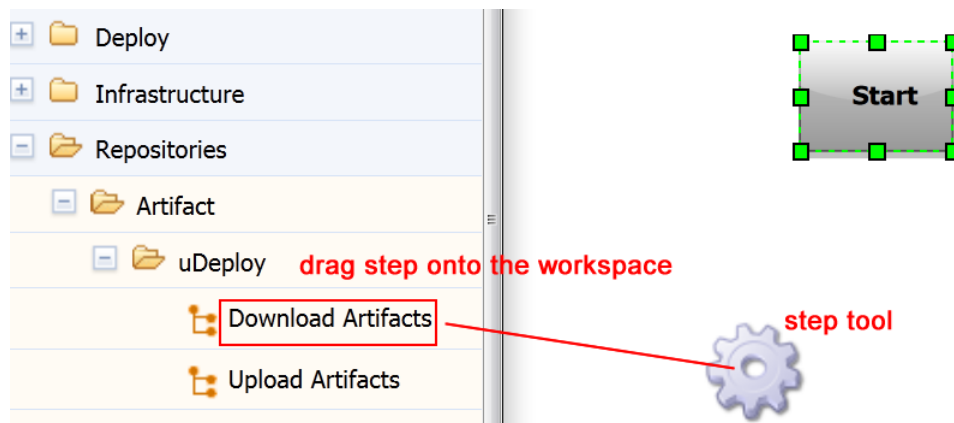
## Adding Process Steps

**To add a step:**

1. In the **Available Plug-in Steps** list, click and hold down the mouse on the step you want to use, and drag it onto the design space.

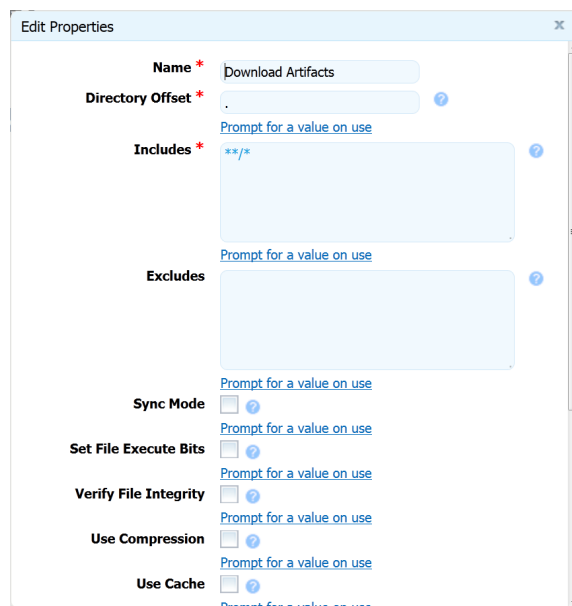
The cursor changes to the *step tool*.



**Figure 26. Adding a Step**

2. Release the step tool over the design space.

The **Edit Properties** pop-up is displayed. Because connections are created after configuring the step's properties, you can place the step anywhere on the design space. Steps can be dragged and positioned at any time. See *Plug-ins* for information about configuring specific steps.

**Figure 27. Typical Edit Properties Pop-up**

Configuration dialogs are tailored to the selected step--only parameters associated with the step type are displayed.

3. After configuring the step's properties, save the step by clicking the **Save** button.

The step is in the design space and ready to be connected to other steps. If you change your mind, click the **Cancel** button to remove the step from the design space. You can add connections immediately after placing a step or place several steps before defining connections.

## Connecting Process Steps

Connections control process flow. The originating step will process before the target step. Creating a connection between steps is a simple process: you drag a connection from the originating step to the target step. Connections are formed one at a time between two steps, the originating step and the target step.

### To create a connection:

1. Hover the cursor over the step that you want to use as the connection's origin.

The connection tool is displayed.

**Figure 28. Connection Tool**

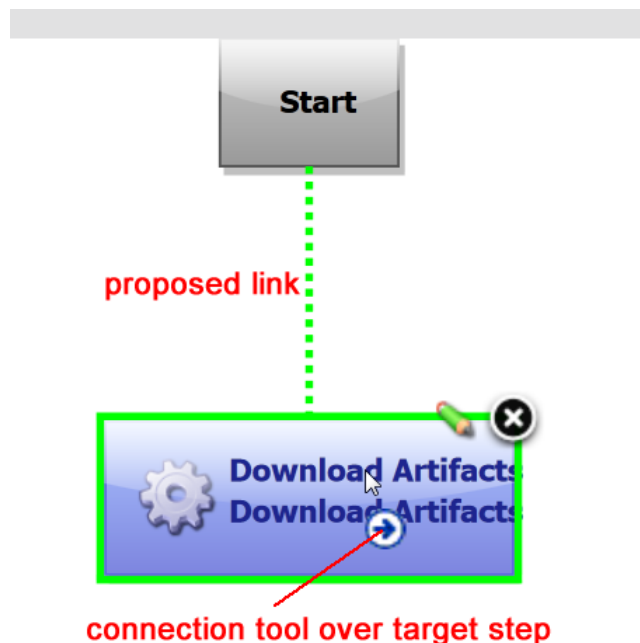
connection tool



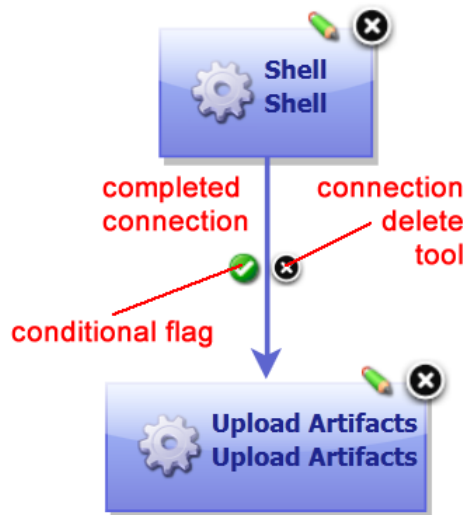
2. Drag the connection tool over the target step.

The step beneath the connection tool is highlighted.

**Figure 29. Dragging the Connection Over a Target Step**



3. Release the connection tool over the target step to complete the connection.

**Figure 30. Completed Connection**

Each connection has a connection delete tool, *conditional flag*, and might have others depending on the originating step. Remove a connection by clicking on the delete tool.

## Process Properties

A processing property is a way to add user-supplied information to a process. A running process can prompt users for information and then incorporate it into the process. Properties are defined with the **Edit Property** dialog.

To define a property:

1. On the **Properties** tab, click the **Add Property** button.

**Figure 31. Edit Properties Dialog**

The screenshot shows the 'Edit Property' dialog box. It has a title bar with 'Edit Property' and a close button. The dialog contains the following fields and controls:

- Name \***: A text input field.
- Description**: A text input field.
- Label**: A text input field.
- Required**: A checkbox.
- Type \***: A dropdown menu with 'Text' selected.
- Default Value**: A text input field.
- Save** and **Cancel** buttons at the bottom.

2. In the **Edit Properties** dialog, enter a name in the **Name** field.
3. Optionally, enter a description in the **Description** field.

4. Enter a label in the **Label** field.

The label will be associated with the property in the user interface.

5. If the property is required, check the **Required** check box.

Default value is unchecked--not required.

6. Specify the type of expected value with the **Type** drop-down list box.

Supported types are: *text*, *text area*, *check box*, *select*, *multi select*, and *secure*. Default type is *text*.

7. In the **Default Value** field, enter a default value (if any).

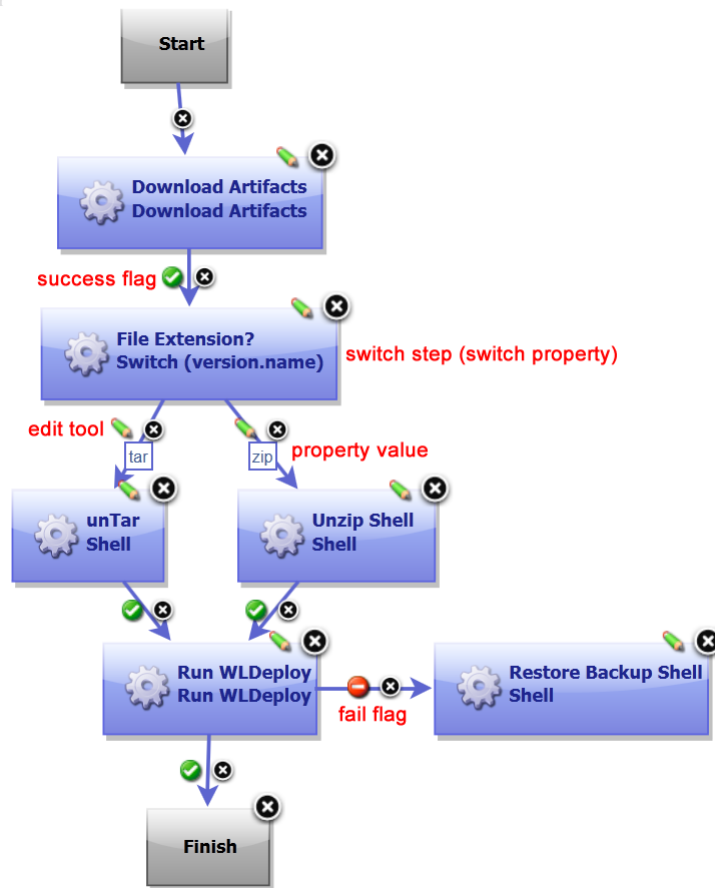
8. To save your work, click the **Save** button. To discard changes, use the **Cancel** button.

To use a property in a process, reference it when you configure (see the section called “Component Processes”) a step that uses it.

## Switch Steps and Conditional Processes

Every connection (except connections from the Start step) has a delete tool and conditional flag. The conditional flag enables you to set a condition on a connection. The condition refers to the processing status of the originating step--success or failure. Possible flag conditions are: *success* (the process completed successfully), *fail* (the process did not finish successfully), or *both* (accept either status). By default, all connections have the flag set to checked (true), meaning the originating step must successfully end processing before the target step starts processing.

To change a flag's value, cycle through possible values by clicking the flag.

**Figure 32. Process with Switch Step**

A *switch step* is a IBM uDeploy-supplied utility step that enables process branching based on the value of a property set on the step. The accompanying figure illustrates a switch step. In this case, the switch property is *version.name*. The connections from the switch step represent process branches dependent on the value of *version.name*. In this example, regardless of which branch is taken, the process will proceed to the *Run WLDeploy* step. Note that *Run WLDeploy* has success and fail conditions.

See *Plug-ins* for information about configuring specific steps.

### Note

If a step has multiple connections that eventually reach the same target step, determining whether the target will execute depends on the value of the intervening flags. If all of the intervening connections have success flags, the target will only process if all the steps are successful. If the intervening connections consist of an assortment of success and fail flags, the target will process the first time one of these connections is used.

For a process to succeed, execution must reach a Finish step. If it does not end with Finish, the process will fail every time.

## Process Step Properties

All steps have the following properties: *exitCode*, *status*, *lines of interest* (LOI—items the post-processing script finds in the step's output log).

You can view the properties by using the component's Log pane to examine the step's output log (*Components > [selected component] > [View Request action] > [Input/Output Properties action]*).

Inventory and versions statuses, which are defined with the *status* property, can be used in application approval gates (see the section called “Application Gates”). The other properties can be used by post-processing scripts, see the section called “The <post-processing> Element”.

## Component Manual Tasks

A component manual task is a mechanism used to interrupt a component process until some manual intervention is performed. A task-interrupted process will remain suspended until the targeted user or users respond. Typically, manual tasks are removed after the process has been tested or automated.

Similar to approvals, triggered tasks alert targeted users. Alerts are associated with component-, environment-, or resource-defined user roles. Affected users can respond—approve—by using the Work Items pane (see the section called “Work Items”). Unlike approvals, manual tasks are incorporated within a component process.

## Creating Component Manual Tasks

To create a task:

1. Display the Create New Task Definition dialog (*Components > [selected component] > Tasks > Create New Task Definition [button]*).
2. Name the task then select a template from the Template Name field.

The individual tasks map to the notification scheme used by the application(see ???). If a scheme is not specified, the default scheme is used. The available tasks are:

- ApplicationDeploymentFailure
- ApprovalCreated
- TaskCreated
- ProcessRequestStarted
- DeploymentReadied
- ApplicationDeploymentSuccess
- Approval Failed

## Using Component Manual Tasks

Component manual tasks are implemented with the Manual Task component process step. Use the step to insert a manual task trigger into a component process.

**Table 13. Component Manual Task Properties**

Field	Description
Name	Typically the name and description correspond to the component process.
Task Definition	Used to select a user-defined task, as described above.
Component Role	Select the role expected to respond. The user mapped to this role will have to respond to the

Field	Description
	generated work item before the process can continue.
Environment Role	Select the role expected to respond. The user mapped to this role will have to respond to the generated work item before the process can continue.
Resource Role	Select the role expected to respond. The user mapped to this role will have to respond to the generated work item before the process can continue.

If multiple roles are selected, all affected users will have to respond before the process can continue. See ??? for information about notification schemes; see the section called “Process Editor” for information about creating component processes.

## Post-Processes

When a plug-in step finishes processing, its default post-processing element is executed. The post-processing element is defined in the plug-in's XML definition, see the section called “Creating Plug-ins”

You can override the default behavior by entering your own script into the step's Post Processing Script field. A post-processing script can contain any valid JavaScript code. Although not required, it's recommended that scripts be wrapped in a CDATA element.

See the section called “The <post-processing> Element” for more information.

## Component Templates

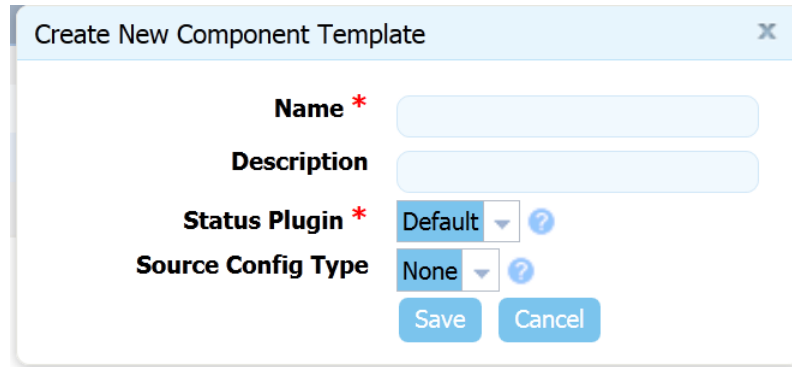
There are two types of templates available:

- A *component template* enables you save and reuse component processes and properties and create new components from them; template-based components inherit the template's properties and process.
- A *configuration template* is typically used to save server or property configurations.

## Creating a Component Template

**To create a template:**

1. Display the Create New Component Template dialog (*Components > Templates > Create New Template [button]*).

**Figure 33. Create New Component Template Dialog**


The dialog box titled "Create New Component Template" contains the following fields and controls:

- Name \***: A text input field.
- Description**: A text input field.
- Status Plugin \***: A dropdown menu with "Default" selected and a help icon (?) to its right.
- Source Config Type**: A dropdown menu with "None" selected and a help icon (?) to its right.
- Buttons**: "Save" and "Cancel" buttons at the bottom right.

2. Enter the template's name in the **Name** field.

3. Enter a description in the **Description** field.

The optional description can be used to convey additional information about the template.

4. Select a plug-in from the **Status Plug-in** field.

If you previously created any status-related plug-ins, they will be listed here. The default value is *Default*, meaning that the template will have IBM uDeploy-supplied steps available for use.

5. Select the source for the artifacts from the **Source Config Type** drop-down list.

Selecting a value other than the default *None*, displays additional fields associated with your selection. Source-dependent fields are used to identify and configure the artifacts. If you select a source, components based on the template will use the same source. See *Component Source Configuration*

### Note

If you select a source, any properties you configure will be set for any components created with the template.

6. Click the **Save** button to save the template.

Saved templates are listed in the **Component Templates** pane.

You create a process for the template in the same way processes are created for components. For information about creating component processes, see the section called "Process Editor".

## Importing\Exporting Templates

Templates can be imported and exported.

### Exporting Templates

Exporting a template creates a JSON file (file extension `.json`) that contains the template's configuration information, properties, and processes.

**To export a template:**



On the Component Templates pane (Components > Templates), click the Export link in the Actions field. You can load the file into a text editor, or save it. If you save it, a file is created with the same name as the selected component, for example, `helloWorldTemplate.json`.

## Importing Templates

When you import a template, you can create an entirely new template or upgrade an existing one.

### To import a template:

1. Display the Import Template dialog (Components > Templates > Import Template [button]).
2. Enter the path to the JSON file containing the template or use the Browse button to select one.
3. If you want to upgrade an existing template, check the Upgrade Template check box. To create a new template, leave the box unchecked.

If the template's name in the JSON file (not the name of the file itself) matches an existing template, the template will be upgraded. If the name is not found, the command has no effect.

### Note

The template's name is the first parameter in the JSON file; for example,

```
"name": "helloWorldTemplate",
```

```
.
```

4. Click Submit.

## Component Template Properties

Component template properties ensure that every component created from a template has the same properties. The three types of available properties are described in the following table.

**Table 14. Component Template Properties**

Type	Description
Properties	Custom property. Every component will inherit the value defined in the template (it cannot be overridden by a component). If you change the value, the change will be reflected in components created from the template, including those previously created.
Component Property Definitions	Every component will have this property; it will appear on the Create New Component dialog for every component created from this template (see the section called “Creating Components”) A value defined here can be changed by created components. Each property must have a type: <ul style="list-style-type: none"> <li>• <i>Text</i></li> </ul> <p>Enables users to enter text characters.</p>

Type	Description
	<ul style="list-style-type: none"> <li>• <i>Text Area</i> Enables users to enter an arbitrary amount of text, limited to limited to 4064 characters.</li> <li>• <i>Check Box</i> Displays a check box. If checked, a value of <i>true</i> will be used; otherwise the property is not set.</li> <li>• <i>Select</i> Requires a list of one or more values which will be displayed in a drop-down list box. Enables a single selection. <i>Note: not currently implemented.</i></li> <li>• <i>Multi Select</i> Requires a list of one or more values which will be displayed in a drop-down list box. Enables multiple selections.</li> <li>• <i>Secure</i> Used for passwords. Similar to Text except values are redacted.</li> </ul>
Environment Property Definitions	<p>Every environment that uses a component created by this template will have this property. The property will appear on the environment's Component Mappings pane (<i>Applications &gt; [selected application] &gt; Environments &gt; [selected environment] &gt; Component Mappings</i>), see the section called “Application Environments”. A value defined here can be changed by environment. Each property must have a type:</p> <ul style="list-style-type: none"> <li>• <i>Text</i> Enables users to enter text characters.</li> <li>• <i>Text Area</i> Enables users to enter an arbitrary amount of text, limited to limited to 4064 characters.</li> <li>• <i>Check Box</i> Displays a check box. If checked, a value of <i>true</i> will be used; otherwise the property is not set.</li> <li>• <i>Select</i></li> </ul>

Type	Description
	<p>Requires a list of one or more values which will be displayed in a drop-down list box. Enables a single selection. <i>Note: not currently implemented.</i></p> <ul style="list-style-type: none"><li>• <i>Multi Select</i></li></ul> <p>Requires a list of one or more values which will be displayed in a drop-down list box. Enables multiple selections.</p> <ul style="list-style-type: none"><li>• <i>Secure</i></li></ul> <p>Used for passwords. Similar to Text except values are redacted.</p>

## Using Component Templates

When you create a component based on a template, the component inherits the template's process (if any, see the section called “Component Processes”), and properties (if any, the section called “Creating Components”).

### To create a template-based component:

1. Display the Create New Component dialog (*Components > Templates > [selected template] > Create New Component [button]*).

The Create New Component dialog (the same dialog used to create non template-based components) is used to configure component. Properties defined in the template will be predefined. If a source was selected in the template, the source is set here and the Source Config Type field is locked. For information about using this dialog, see the section called “Creating Components”

2. After configuring editable properties, save the component.

Templates used to create components are listed in the Templates view.

Components created from templates are listed in the Components view.

## Configuration Templates

Configuration templates, as the name implies, contain configuration data. Typically, the data is for server configurations Tomcat servers, for instance, but the data can be for any purpose.

### To create a configuration template:

1. Display the Create New Configuration Template dialog (*Components > [selected component] > Templates > Create New Configuration Template [button]*).

**Figure 34. Create New Configuration Template Dialog**

**Create New Configuration Template**

**Name \*** tomEuro

**Template**

```

locked/agent.keystore=../conf/ud.keystore
locked/agent.http.proxy.port=
locked/agent.jms.remote.port=@AGENT_REMOTE_PORT@
locked/agent.home=@AGENT_HOME@
locked/agent.brokerUrl=failover\:(ah3\://@AGENT_REMOTE_HOST@
\:@AGENT_REMOTE_PORT@)
locked/agent.mutual_auth=false
locked/agent.name=@AGENT_NAME@
UrbanDeploy/java.home=@JVM_HOME@
locked/agent.http.proxy.host=
locked/agent.keystore.pwd=pbe{7mAgcNtask3FXS56P1igcqcw95zwh
/VeNBa9FGb2Slw\=}
locked/agent.jms.remote.host=@AGENT_REMOTE_HOST@

```

Save Cancel

2. Enter a meaningful name in the Name field.
3. In the Template field, enter or paste the template text. Text can be in any script—or no script at all. The amount of text is based on the database used by IBM uDeploy. Practically there is no limit to the amount of text used for a configuration template.
4. Save your work when you are finished.

Configuration templates can be edited at any time by using the Edit action.

## Component Change Logs

Change logs provide information about modifications to components. To see change details, display the log for a selected component-related activity (*Home > Components > Changes [selected component] > Changes > Changes [action for selected item].*). Information for any change that triggered a Commit ID is displayed.

## Deleting and Deactivating Components

Components can be deactivated and deleted. To delete or deactivate a component, use the desired action on the Components pane for the intended component.

When a component is deactivated, it remains in the database and CodeStation and can be activated later. To activate a component, first click the Show Inactive Components check box, then use the Activate action for the component.

When a component is deleted, it, along with all version, is removed from the database and CodeStation and cannot be activated at a later time (the original artifacts are not affected—only CodeStation copies are deleted). Components cannot be deleted if they are used by an application. To delete a component used by an application, first remove the component from the application.

# Resources

To run a deployment, IBM uDeploy requires an agent (resource) or proxy agent on the target machine. Typically, an agent is installed in every environment that an application passes through. A typical production pipeline might be, say, SIT, UAT, PROD (the application passes through two testing environments before reaching production). In this scenario, at least three agents need to be installed--one per environment. If different components run on different machines within a given environment, you might want to install multiple agents in that environment.

Whether you need one or multiple resources per environment is determined by your current infrastructure, deployment procedures, and other requirements. Many IBM uDeploy users have significant differences among environments--in SIT you might need to deploy a component to one machine, while in UAT you might need to deploy the component to multiple machines. You could, for example, configure sub-groups for the single agent in the SIT environment and then set up individual resources for each agent in the UAT environment.

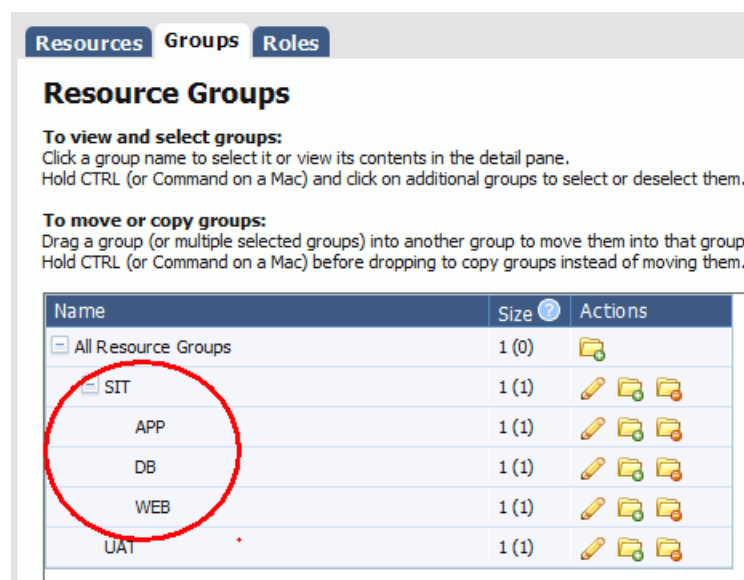
## Resource Groups

IBM uDeploy uses the concept of resource groups to help you organize and manage the agents installed in different environment throughout the network. You need to create at least one resource group per installed agent, as when configuring your Processes you will need to select the appropriate Group. What groups you create and how you organize the groups, e.g., using subgroups, depends on your existing organizational processes and infrastructure.

### Note

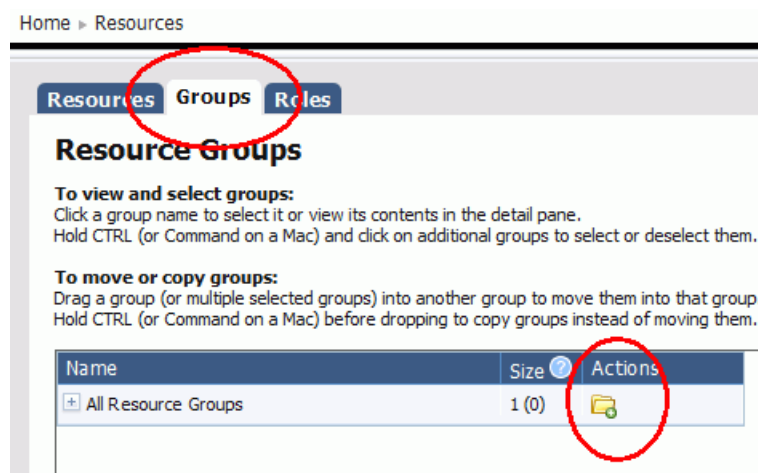
Before continuing, ensure that at least one agent has been installed in a target environment (for evaluation purposes, the agent can be on the same machine as the server).

**Figure 35. Groups**



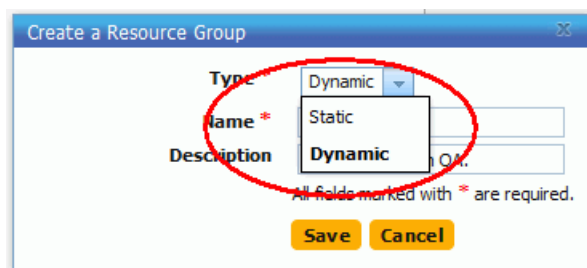
## Creating a Resource Group

1. Go to *Resources > Groups*. and click on the folder icon.

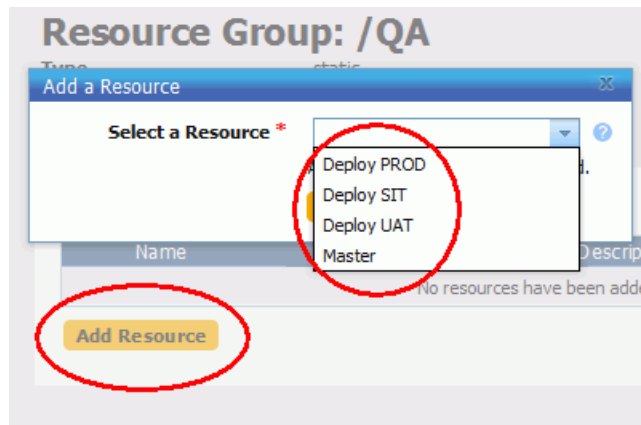
**Figure 36. Action Tool**

2. For the Type, most often Static is used.

Name and description. Typically, the name will correspond to either the Environment the Resource participates in, the Component that uses the Resource Group, or a combination of both (e.g., SIT, DB, or SIT-DB). What description you give depends on how you intend to use the Resource that this Group is assigned to, etc.








**Figure 37. Create a Resource Group Dialog**

3. Once the Resource has been created, select the pencil icon to edit the Group.

**Figure 38. Add a Resource Dialog**

4. Once you assign a Group to a Resource, you add Subresources. A subresource enables you to apply logical identifiers, or categories, within any given Group. During deployment configuration, you can Select a given Subresource that the Process will run on. To create a Subresource, select the New Resource icon for the Group. Configuration is similar to Resource Group creation.

**Figure 39. Sub-resources**

Name	Size	Actions
All Resource Groups	1 (0)	
PROD	0 (0)	  
QA	1 (1)	  

## Resource Roles

A role enables you to further refine how a resource is utilized, and is similar to sub resources. For most deployments, you will not need to define a role. During process configuration, you select a specific role when determining the resource. A role can be used to set up IBM uDeploy for rolling deployments, balancing, etc. For example, you can set up your process to only deploy to a percentage of targets first; add a manual task in the middle of the process that requires a user to execute (e.g., after they have tested the partial deployment); and then once the manual task has completed the rest of the process is assigned a second role responsible for deploying to the rest of the target machines.

## Role Properties

When you create a role, you can define properties for it then whenever you add the role to a resource, you can set the values for the properties. For example, if you create a role called "WS" and define a property call "serverURL," you can access the property like this: `${p:resource/WS/serverURL}`. For information about IBM uDeploy properties, see *IBM uDeploy Properties*

## Agents

An agent is a lightweight process that runs on a target host and communicates with the IBM uDeploy server. Agents perform the actual work of deploying components and so relieves the server from the task, making large deployments involving thousands of targets possible. Usually, an agent runs on the same



host where the resources it handles are located; a single agent can handle all the resources on its host. If a host has several resources, an agent process is invoked separately for each one. Depending on the number of hosts in an environment, a deployment might require a large number of agents.

Agents are installed with the batch files provided with the installation files, see the section called “Agent Installation”. Agents that will be installed on Unix machines can also be installed remotely using IBM uDeploy's web application, which is described below. Agents are run using the batch files included with the installation package.

Once an installed agent has been started, the agent opens a socket connection to the IBM uDeploy server (securable by configuring SSL for server-agent communication) based on the information supplied during installation. Agents on networks other than the one where the server is located might need to open a firewall to establish connection. Once communication is established, the agent will be visible in the IBM uDeploy web application where it can be configured. Active agents--regardless of OS--can be upgraded using the web application.

Agent configuration consists of assigning an agent to at least one environment; agents can be assigned to multiple environments. If an agent is assigned to several environments, it can perform work on behalf of all of them.

## Remote Agent Installation

You can install an agent onto a Unix machine using the web application. A remotely installed agent cannot be installed as a service.

To install an agent:

1. Display the **Install New Agent** dialog by clicking the **Install New Agent** button on the **Agents** pane (*Home > Resources > Agents*).
2. Enter the required information into the dialog's fields:

**Table 15. Remote Agent Installation Fields**

Field	Description
Target Hosts*	Host names or IP addresses of the machines where the agent will be installed.
SSH Port*	SSH port addresses of the machines where the agent will be installed.
SSH Username*	SSH user name used on the machines where the agent will be installed.
Use Public Key Authentication	Check this box if you want to authenticate using public key authentication instead of a password.
SSH Password*	SSH password associated with the user name used on the machines where the agent will be installed.
Agent Name*	Name of the agent.
Agent Dir*	Directory where agent should be installed.
Java Home Path*	Path to Java on the machine where the agent will be installed.
Temp Dir Path*	Path to the directory used to perform the installation on the target machine.

Field	Description
Server Host*	Host name or IP address of the IBM uDeploy server or agent relay to which the agent will connect.
Server Port*	IBM uDeploy server port (7918) or agent relay (7916) to which the agent will connect.
Mutual Authentication	Check this box if the agent should enforce certificate validation for mutual authentication.
Proxy Host	Host name or IP address of the agent relay if used.
Proxy Port	HTTP port of the agent relay (20080) if used.

3. Click **Save** when you are done.

Remotely installed agents will start running automatically. If a remotely installed agent stops running, it must be restarted on the host machine.

## Managing Agents Remotely

While we characterize an agent as *a* process (singular), technically an agent consists of two processes: a *worker* process and a *monitor* process. Worker processes perform the actual work of deployment, such as handling plug-in steps. Monitor processes manage the worker process: handling restarts, upgrades, and tests for example. Once an agent is installed, you can manage (via the monitor process) many of its features from the IBM uDeploy web application. Agent properties can be changed directly by editing the agent's `conf/agent/installed.properties` file and restarting the agent.

To manage an agent:

1. Display the **Agents** pane (*Home > Resources > Agents*).
2. Click an action link for the desired agent. Actions are described in the following table.

**Table 16. Agent Management**

Action	Description
Edit	This option enables you to edit the agent's description.
Restart	This option will shutdown and restart the agent. While the agent is shutdown, its status will be <i>Offline</i> .
Upgrade	This option will shutdown the agent and apply the upgrade. While the agent is shutdown, its status will be <i>Offline</i> . After the upgrade is applied, the agent will be restarted. Before its status is <i>Online</i> , it might briefly be <i>Connected</i> .
Test	This option will perform an agent settings and connection test. Test results are displayed in the <b>Connection Test</b> dialog.
Inactivate	This option will deactivate the agent. Agents that are deactivated cannot perform deployments.

Action	Description
	To reactivate the agent, check the <i>Show Inactive Agents</i> check box on the <b>Agents</b> pane, then click <i>Activate</i> for the agent.
Delete	Removes the agent.

## Agent Pools

Similar to resource groups, agent pools help you organize and manage agents installed in different environments.

### Creating an Agent Pool

To create an agent pool:

1. Display the **Create New Agent Pool** dialog by clicking the **Create New Agent Pool** button on the **Agent Pools** pane (*Home > Resources > Agent Pools*).
2. Enter the pool name in the *Name* field.
3. Optionally, enter a description in the *Description* field.
4. Click the *Pool Members* field to add agents to the pool. A selection-type pop-up is displayed listing the available agents.
5. Select the agent or agents you want to add to the pool. Optionally, you can filter the listed agents by entering search text into the text field.
6. When you are finished, click **Save**.

### Managing Agent Pools

To manage agent pools:

1. Display the **Agent Pools** pane (*Home > Resources > Agent Pools*).
2. Click an action link for the desired pool. Actions are described in the following table.

**Table 17. Agent Pool Management**

Action	Description
Edit	This option enables you to add/remove agents and edit the pool's name and description.
Copy	Copies (creates a new pool with the same agents as the selected pool) the pool.
Inactivate	This option will deactivate the agent pool.
Delete	Removes the agent pool.

---

# Applications

Applications are responsible for bringing together all the components that need to be deployed together. This is done by defining the different versions of each component as well as defining the different environments the components must go through on the way to production. In addition, Applications also map the constituent hosts and machines (called resources) a component needs within every environment.

Applications also implement automated deployments, rollbacks, etc. These are called Processes; however, at the Application level Processes are only concerned with the Components and Resources necessary for deployment, etc. -- differentiating Application-processes from those of Components (which are concerned with running commands, etc.).

Applications also introduce Snapshots to manage the different versions of each Component. A snapshot represents the current state of an Application in the Environment. Typically, the Snapshot is generated in an Environment that has no Approval gates -- called an uncontrolled Environment. For most users, the Snapshot is pushed through the pipeline.

## Note

Before configuring an Application, you will need to ensure that at least one agent has been installed in a target environment (for evaluation purposes, the agent can be on the same machine as the server). In addition, you will also need to add at least one Resource Group to the agent. See *Resources*.

## Environments

An Environment is a collection of Resources that host the Application. Environments typically include host machines and IBM uDeploy agents. When a deployment is run, it is always done so in an Environment. While Environments are collections of Resources, Resources can vary per Environment.

For example, Environment 1 may have a single web server, a single middleware server, and a single database server, that must be deployed to; IBM uDeploy represents these as three, separate Resources running in Environment 1. Environment 2, however, may have a cluster of Resources that the same Application must be deployed to. IBM uDeploy compensates for these differences with Resource Groups (more at Resources by keeping an Inventory of everything that is deployed to each Environment: IBM uDeploy knows exactly the Environment and Server(s) where the Application was deployed to: and tracks the differences between the Environments.

## Processes

A process plays a coordination role. They are authored using a visual drag-n-drop editor, and composed of Steps that call the Component Processes. For example, to deploy the Application you may invoke a Process called Deploy. This Deploy Process would in turn call out to the requisite Components and execute the deployment.

## Snapshots

Snapshots specify what combination of Component versions you deploy together. They are models you create before deploying the Application. A Snapshot specifies the exact version for each Component in the Application. When a Snapshot is created, IBM uDeploy gathers together information about the Application, including the Component versions, for a given Environment. Typically, the Snapshot is generated in an Environment that has no Approval gates -- called an uncontrolled Environment. For most users, the Snapshot is pushed through the pipeline. Typically, one of the Environment will always remain uncontrolled to allow for Snapshots. When a successful deployment has been run in the uncontrolled

Environment, a Snapshot is created based on the Application's state within the Environment: thus capturing the different versions of the Components at that time. As the Application moves through various testing Environments, for example, IBM uDeploy ensures that the exact versions (bit for bit) are used in every Environment. Once all the appropriate stages and Approvals for a Snapshot are complete, the Snapshot is pushed to Production.

## Creating Applications

You can create an application from scratch or import an existing one. See the section called “Importing/Exporting Applications” for information about importing applications. After creating an application, you:

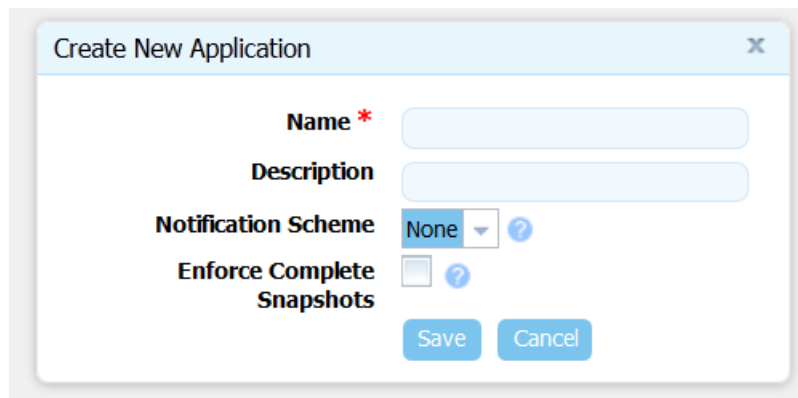
- add components (the section called “Adding Components to an Application”)
- create an environment (the section called “Creating an Environment”)
- associate an agent with the environment (the section called “Mapping Resources to an Environment”)
- create an application process (the section called “Application Processes”)

Before configuring an application, ensure that at least one agent has been installed in a target environment (for evaluation purposes, the agent can be on the same machine as the server). See *Resources*.

### To create an application:

1. Display the Create New Application dialog *Applications > Create New Application [button]*, and enter the following:

**Figure 40. Create New Application Dialog**



- Typically the name and description correspond to the application you plan on deploying.
- Notification Scheme. IBM uDeploy includes integrations with LDAP and e-mail servers that enable it to send out notifications based on events. For example, the default notification scheme will send out an e-mail when an application deployment fails or succeeds. Notifications also play a role in approving deployments: IBM uDeploy can be configured to send out an e-mail to either a single individual or to a group or people (based on their security role) notifying them that they need to approve a requested deployment. See ???.
- If you want the application to require that every component is versioned, click the Enforce Complete Snapshots check box.

2. Save your work when done.

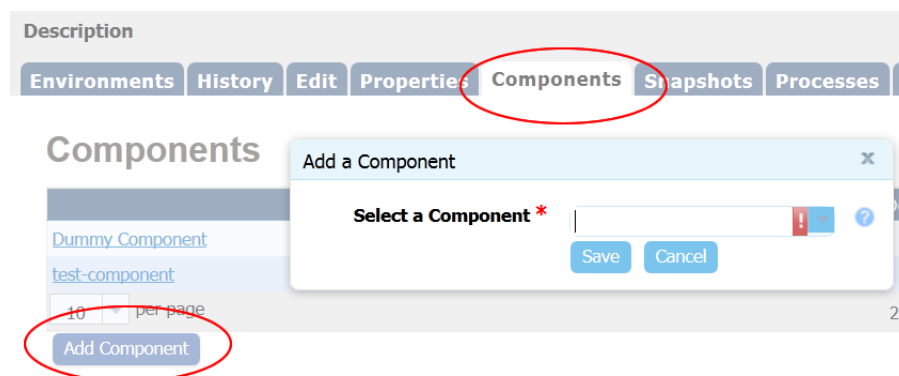
## Adding Components to an Application

Next, add at least one component to the application. Applications bring the different components (their versions and processes) together so they can be deployed as a single unit.

**To add components to an application:**

1. Display the Add a Component dialog *Applications* > [*select application*] > *Components* > *Add Component* [button]

**Figure 41. Selecting a Component**



2. Use the Select a Component list box to choose a component. Components are selected one at a time.

## Importing/Exporting Applications

Applications can be imported and exported. Importing/exporting can be especially useful if you have multiple IBM uDeploy servers, for example, and need to quickly move or update applications.

### Exporting Applications

Exporting an application creates a JSON file (file extension `json`) that contains the application's properties, components (and their associated properties and processes), and processes. For information about JSON, see <http://www.json.org/>.

**To export an application:**

On the Applications pane (Home > Applications), click the **Export** link in the Actions field. You can load the file into a text editor, or save it. If you save it, a file is created with the same name as the selected component, for example, `helloWorldApplication.json`.

### Importing Applications

When you import an application, you can create an entirely new application or upgrade an existing one. Components—including their properties and processes—associated with the application are also imported (if available to the importing server). For information about templates associated with imported components, see the section called “Importing/Exporting Components”.

## Note

If imported components have the Import Versions Automatically parameter set to true, IBM uDeploy will automatically import component versions as long as the artifacts are accessible to the importing server.

## To Import an Application

1. Display the Import Application dialog (Applications > Import Application [button]).
2. Enter the path to the JSON file containing the application definition or use the Browse button to select one.
3. If you want to upgrade an existing application, check the Upgrade Application check box. To create a new application, leave the box unchecked.

If the application's name in the JSON file (not the name of the file itself) matches an existing application, the application's parameters are updated with new values, and new items—such as processes, environments, and components—are added. If the name is not found, the command has no effect.

## Note

The application's name is the first parameter in the JSON file; for example,

```
"name": "helloWorldApplication",  
.
```

4. Specify how imported components should be handled with the Component Upgrade Type drop-down box. For these options, the components must be on the importing server.
  - To use the same components used by the imported application, select Use Existing Component. The new application will contain references to the imported applications components. This option is especially useful if you are importing a lot of applications.

If you are upgrading, the application will use the imported components, and no longer use any not used by the imported application.

- To create new components based on those used by the imported application, select Create New Component. New components will be created (based on the imported application's components).

If you are upgrading, the application will use the newly created components and no longer use any it previously used.

- When you want to create a fresh installation, select Fail if Component Exists. If you are creating an application, it will create both a new application and component unless the component already exists, in which case the application is not imported.

If you are upgrading, the upgrade will fail if any imported components already exist on the importing server.

- To ensure a component is on the importing server, select Fail if Component Does Not Exist. If you are creating an application, it will create both a new application and component unless the component does not exist, in which case the application is not imported.

If you are upgrading, the upgrade will fail if an imported component does not already exist on the importing server.

- To upgrade existing components, select Upgrade if Exists. This option creates an application and upgrades existing components with data from the imported application.

If you are upgrading and existing components match imported ones (all must match), the components will be upgraded. If none of the imported components match existing ones, the imported components will be used.

5. Click Submit.

## Application Environments

An environment is a user-defined collection of resources that hosts an application. An environment is the application's mechanism for bringing together components with the agent that actually deploys them. Environments are typically modeled on some stage of the software project life cycle, such as development, QA, or production. A resource is a deployment target, such as a database or J2EE container. Resources are usually found on the same host where the agent that manages them is located. A host can be a physical machine, virtual machine, or be cloud-based.

Environments can have different topologies—for example: an environment can consist of a single machine; be spread over several machines; or spread over clusters of machines. Environments are application scoped. Although multi-tenant machines can be the target of multiple applications, experience has shown that most IT organizations use application-specific environments. Additionally, approvals are generally scoped to environments.

IBM uDeploy maintains an inventory of every artifact deployed to each environment and tracks the differences between them.

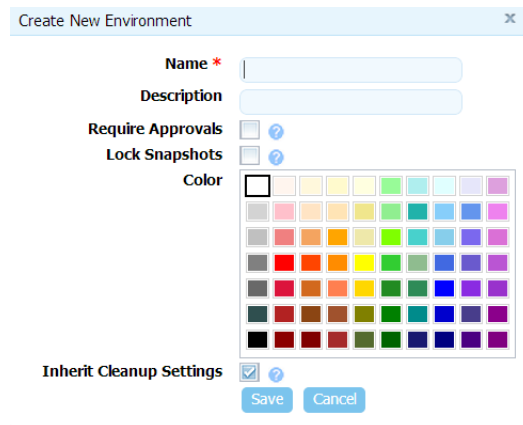
## Creating an Environment

Before you can run a deployment, you must define at least one environment that associates components with an agent on the target host. This initial environment is typically uncontrolled and often used to create snapshots.

**To create an environment:**

1. Display the Create New Environment dialog *Applications > [select application] > Environments > Add New Environment [button]* , then enter the following:



**Figure 42. Create New Environment dialog**


The dialog box titled "Create New Environment" contains the following fields and controls:

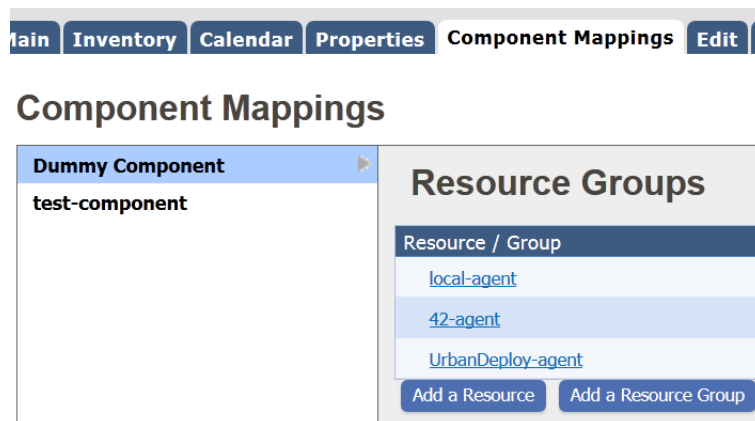
- Name \***: A text input field.
- Description**: A text input field.
- Require Approvals**: A checkbox with a help icon.
- Lock Snapshots**: A checkbox with a help icon.
- Color**: A color picker grid showing various color swatches.
- Inherit Cleanup Settings**: A checked checkbox with a help icon.
- Buttons**: "Save" and "Cancel" buttons at the bottom right.

- Name and Description. The name is used as part of the deployment process and typically corresponds to the target environment. For example, if you are deploying to an integration environment, "SIT" might be appropriate.
- To ensure that components cannot be deployed to the environment without first being approved, click the Require Approvals check box. If checked, IBM uDeploy will enforce an approval process before the deployment can be deployed to the environment. Initial deployments are typically done in uncontrolled environments, but once the deployment is successful, you can configure an approvals process as the application moves along the development pipeline. If you are setting up more than one environment, consider creating an approvals process for at least one of them.
- If the Lock Snapshots check box is selected, snapshots added to this environment will be locked (for the selected environment) to prevent changes.
- The Color picker enables you to apply a visual identifier to the environment. The selected color will appear in the UI.
- The Inherit Cleanup Settings check box determines how many component versions are kept in CodeStation, and how long they are kept. If checked, the application will use the values specified on the System Settings pane. If unchecked, the Days to Keep Versions (initially set to -1, keep indefinitely) and Number of Versions to Keep (initially set to -1, keep all) fields are displayed, which enable you to define custom values. The default value is checked.

2. Save your work when you are done.

## Mapping Resources to an Environment

1. After you have added a component to the application, define where its artifacts should be deployed by selecting a resource (agent) or resource group. See *Resources*.
1. Display the Component Mappings pane (*Applications* > *[selected application]* > *Environments* > *[selected environment]* > *Component Mappings*).

**Figure 43. Component Mapping**

2. If the application has several components associated with it, select the one you want to use from the component list. Each component associated with this application can be mapped to a different agent (resource).
3. To associate a resource with the selected component:
  - To add a resource group, click the Add a Resource Group button and select a resource group. For information about creating resources, see the section called “Resource Groups”.
  - To add a resource, click the Add a Resource button and select an resource.

After mapping components and resources, make the application deployment ready by creating an application process, which is described in the following section.

## Environment Properties

Environment properties can be created with the environment's Properties pane (*Applications* > *[selected application]* > *Environments* > *[selected environment]* > *Properties*).

A value set on component environment overrides one with the same name set directly on an environment property. Component environment properties enable you to centralize properties, tracking type and default values, for instance. Environment properties provide ad-hoc lists of `property=value` pairs.

Referenced: `${p:environment/propertyName}`.

## Application Processes

Application processes, like component processes, are created with the process editor. IBM uDeploy provides several common process steps, otherwise application processes are assembled from processes defined for their associated components.

Application processes can run manually, automatically on some trigger condition, or on a user-defined schedule. When a component has several processes defined for it, the application determines which ones are executed and in which order.

An application process is always associated with a target environment. When an application process executes, it interacts with a specific environment. At least one environment must be associated with the

application before the process can be executed. Application processes are environment agnostic; processes can be designed independently of any particular environment. To use the same process with multiple environments (a typical scenario), you associate each environment with the application and execute the process separately for each one.

In addition to deployments, several other common processes are available, such as rolling-back deployments. IBM uDeploy tracks the history of each component version, which enables application processes to restore environments to any desired point.

## Creating Application Processes

1. Display the Create an Application Process dialog (*Applications > [select application] > Create New Process [button]*), and enter the following information:

Name and Description. Typically the name and description correspond to the application you plan on deploying.

**Figure 44. Create New Application Dialog**

**Table 18. Application Process Fields**

Field	Description
Name/Description	Typically the name and description correspond to the application you plan on deploying.
Required Application Role	Use this drop-down list box to select the role a user must have in order to run the application. For information about creating application roles, see ???. The default value is <i>None</i> .
Inventory Management	If you want to handle inventory manually, select <i>Advanced</i> . To have inventory handled automatically, leave the default value, <i>Automatic</i> , selected.
Offline Agent Handling	Specify how the process reacts if expected agents are offline: <ul style="list-style-type: none"> <li>• Check Before Execution: checks to see if expected agents are on line before running the</li> </ul>

Field	Description
	<p>process. If agents are off line, the process will not run.</p> <ul style="list-style-type: none"> <li>• Use All Available; Report Failure: process will run as long as st least one agent defined in the environment is on line; reports any failed deployments due to off line agents. Useful for rollbacks or configuration deployments.</li> <li>• Always Report Success: process will run as long as st least one agent defined in the environment is on line; reports successful deployments.</li> </ul>

2. Save your work.

Application process—the steps comprising them—are configured with the process editor. For information about using the process editor, see the section called “Process Editor”. For information about individual process steps, see the section called “Application Process Steps”.

## Application Process Steps

Application processes, like component processes, are created with the process editor. IBM uDeploy provides several common process steps, otherwise application processes are assembled from processes defined for their associated components.

## Application Process Steps Details

The application process steps are described in the following topics.

### Finish

Ends processing. A process can have more than one Finish step.

### Install Component

Installs the selected component using one of the processes defined for the component.

**Table 19. Install Component Properties**

Field	Description
Name	Can be referenced by other process steps.
Component	Component used by the step; a step can affect a single component. All components associated with the application are available. If you want to install another component, add another install step to the process.
Use Versions Without Status	Restricts the components that can be used by the step—components with the selected status are

Field	Description
	ignored. Available statuses: <i>Active</i> means ignore components currently deployed; <i>Staged</i> means ignore components currently in pre-deployment locations.
Component Process	Select a process for the component selected above. All processes defined for the component are available. Only one process can be selected per step.
Ignore Failure	When selected, the step will be considered to have run successfully.
Limit to Resource Role	User-defined resource role the agent running the step must have.
Run on First Online Resource Only	Instead of being run by all agents mapped to the application, the step will only be run by the first online agent identified by IBM uDeploy. The mechanism used to identify the "first" agent is database-dependent (thus indeterminate).
Precondition	A JavaScript script that defines a condition that must exist before the step can run. The condition must resolve to true or false.

## Uninstall Component

Uninstalls the selected component.

**Table 20. Uninstall Component Properties**

Field	Description
Name	Can be referenced by other process steps.
Component	Component used by the step; a step can affect a single component. All components associated with the application are available. If you want to install another component, add another install step to the process.
Remove Versions With Status	Restricts the components that are affected by the step, only components with the selected status are affected. Available statuses: <i>Active</i> means use components currently deployed; <i>Staged</i> means use components currently in pre-deployment locations.
Component Process	Select a process for the component selected above. All processes defined for the component are available. Only one process can be selected per step.
Ignore Failure	When selected, the step will be considered to have run successfully.
Limit to Resource Role	User-defined resource role the agent running the step must have.

Field	Description
Run on First Online Resource Only	Instead of being run by all agents mapped to the application, the step will only be run by the first online agent identified by IBM uDeploy. The mechanism used to identify the "first" agent is database-dependent (thus indeterminate).
Precondition	A JavaScript script that defines a condition that must exist before the step can run. The condition must resolve to true or false.

## Rollback Component

Rolls-back a component version; replaces a component version with an earlier one.

**Table 21. Rollback Component Properties**

Field	Description
Name	Can be referenced by other process steps.
Component	Component used by the step; a step can affect a single component. All components associated with the application are available. If you want to install another component, add another install step to the process.
Remove Versions With Status	Restricts the components that are affected by the step, only components with the selected status are affected. Available statuses: <i>Active</i> means use components currently deployed; <i>Staged</i> means use components currently in pre-deployment locations.
Component Process	Select a process for the component selected above. All processes defined for the component are available. Only one process can be selected per step.
Ignore Failure	When selected, the step will be considered to have run successfully.
Limit to Resource Role	User-defined resource role the agent running the step must have.
Rollback type	Determines the type of rollback. Available statuses: <i>Remove Undesired Incremental Versions</i> and <i>Replace with Last Deployed</i> .
Run on First Online Resource Only	Instead of being run by all agents mapped to the application, the step will only be run by the first online agent identified by IBM uDeploy. The mechanism used to identify the "first" agent is database-dependent (thus indeterminate).
Precondition	A JavaScript script that defines a condition that must exist before the step can run. The condition must resolve to true or false.

## Manual Application Task (Utility)

A manual task is a mechanism used to interrupt an application process until some manual intervention is performed. A task-interrupted process will remain suspended until the targeted user or users respond. Typically, manual tasks are removed after the process has been tested or automated.

Similar to approvals, triggered tasks alert targeted users. Alerts are associated with environment- or application-defined user roles. Affected users can respond—approve—by using the Work Items pane (see the section called “Work Items”). Unlike approvals, manual tasks can be incorporated within an application process.

The task used to configure this step must have been previously defined with the Create New Task Definition dialog.

**Table 22. Manual Application Task Properties**

Field	Description
Name	Typically the name and description correspond to the application.
Task Definition	Used to select a user-defined task.
Environment Role	Select the role expected to respond. The user mapped to this role will have to respond to the generated work item before the process can continue.
Application Role	Select the role expected to respond. The user mapped to this role will have to respond to the generated work item before the process can continue.

If both roles are selected, all affected users will have to respond before the process can continue. See ???.

## Application Manual Tasks

A manual task is a mechanism used to interrupt an application process until some manual intervention is performed. A task-interrupted process will remain suspended until the targeted user or users respond. Typically, manual tasks are removed after the process has been tested or automated.

Similar to approvals, triggered tasks alert targeted users. Alerts are associated with environment- or application-defined user roles. Affected users can respond—approve—by using the Work Items pane (see the section called “Work Items”). Unlike approvals, manual tasks can be incorporated within an application process.

## Creating Application Manual Tasks

To create a task:

1. Display the Create New Task Definition dialog (*Applications > [selected application] > Tasks > Create New Task Definition [button]*).
2. Name the task then select a template from the Template Name field.

The individual tasks map to the notification scheme used by the application(see ???). If a scheme is not specified, the default scheme is used. The available tasks are:

- ApplicationDeploymentFailure
- ApprovalCreated
- TaskCreated
- ProcessRequestStarted
- DeploymentReadied
- ApplicationDeploymentSuccess
- Approval Failed

## Using Manual Tasks

Manual tasks are implemented with the Manual Application Task process step. Use the step to insert a manual task trigger into an application process.

## Approval Process

An approval process enables you to define the job that needs approved and the role of the approver. An approval process must be created if the Requires Approval check box is selected when creating/editing an environment. If a scheduled deployment requiring approval reaches its start time without approval given, the process will not run and act as a rejected request. To resubmit a request, you must request a new process. If an approval-requesting process does not have a scheduled deployment time, the process will remain idle until a response has been made.

### Creating an Approval Process

To create an approval process, display the Approval Process Design Pane.

```
(Home>Applications>Application_Name>Environments>Environment:Environment_Name>Approval Process)
```

Once the pane is displayed, select the steps that need approval from the process editor. The steps are based on job type and the role of the approver. You have the option of selecting three job types: the Application, Component, and/or Environment. For help using the process editor see the section called “Process Editor”.

### Reviewing Status

To view the status of the request, display the Deployment Detail pane on the Reports tab. If a request has been approved it will display as success. However, if the request was rejected it will show failed. If a request is failed display the Application Process Request by clicking view request.

If a comment has been made regarding the process, you can view it by clicking the log button in the actions column on the Application Process Request.

## Work Items

If a job requiring approval is created, an approval process will have to be created. The job requiring approval will display in the approvers Work Items tab. Until approved, the job will remain idle if unscheduled. If time has elapsed on a scheduled job needing approval, the job will fail. This control allows the approver to verify the deployment details, and choose the time it is deployed. Notifications are sent to users who are eligible to complete an approval step if the system is configured with an email server and the user has an email address set.

### View Details of Process

In the Works Items tab, the approver can view the name of the process, when the request was submitted, who requested the process, and the snapshot or version used. The approver can also view details of the



environment or resource by clicking the link in the Environment/Resource column. They can view the details of the target by clicking the link in the target column. Or view details on the request by selecting the View Request in the Actions column. The Actions column is also where the approver can respond to the request.

### Responding to Request

To respond to a request, display the Respond dialog box by clicking Respond in the Actions column. The approver has the option of leaving a comment. If a request is rejected the process will not run. If approved, the process will begin.

## Snapshots

A snapshot is a collection of specific component versions and processes, usually versions that are known to work together. Typically, a snapshot is created when a successful deployment has been run in an uncontrolled environment. Snapshots can be created in a controlled environments as well. As the application moves components through various environments, IBM uDeploy ensures that the exact versions and processes you selected are used in every environment. Snapshots help manage complex deployments--deployments with multiple environments and development teams.

## Creating Snapshots

To create a snapshot, display New Application Snapshot pane (Home > Application > Snapshots > Create New Snapshot).

1. Enter the name of your snapshot in the Name field.
2. In the Process Version Locking field, specify how you want IBM uDeploy to select component processes:
  - **Always use Latest Version** Use the most recently defined component process version for each component in the application (default).
  - **Lock to Current Versions** Use the current component process version for each component.
3. For each component in the application, you can specify which version to use:
  - **Add Version** Enables you to select any version in Codestation for the component.
  - **Copy From Environment** Uses the currently deployed (in this environment) component version.
  - **Remove All** Removes all deployed component versions from this environment.
4. Instead of specifying a version for each component, you can use the most recently deployed version (in this environment) for each component in the application by using the Copy All From Environment button.

If you want to discard any selected component versions, use the Clear All Components button.

## Application Gates

Gates provide a mechanism to ensure that component versions cannot be deployed into environments unless they have the gate-specified status. Version statuses are user-defined values that can be applied to component versions and used in component processes or application gates. Version statuses can be

applied through the user interface (*Components* > *[selected component]* > *Versions* > *[selected version]* > *Add a Status* [button]), or by the Add Status to Version plug-in step. They are displayed in the Latest Status field on the component's Versions pane (*Components* > *[selected component]* > *Versions*).

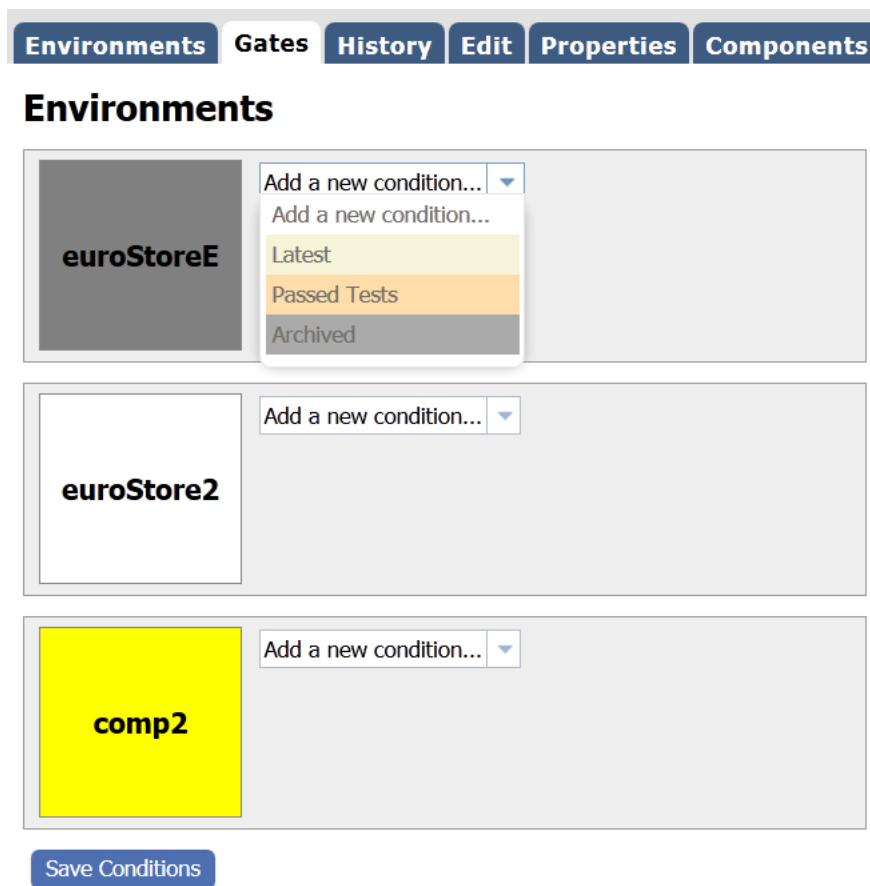
Component version statuses are defined on the Statuses tab (*Settings* > *Statuses*), see ???. Component versions do not have to have gates. Gates are defined at the environment level; an environment can have a single gate defined for it.

## Creating Gates

To create a gate:

1. Display the Gates pane for the target application (*Applications* > *[selected application]* > *Gates*).

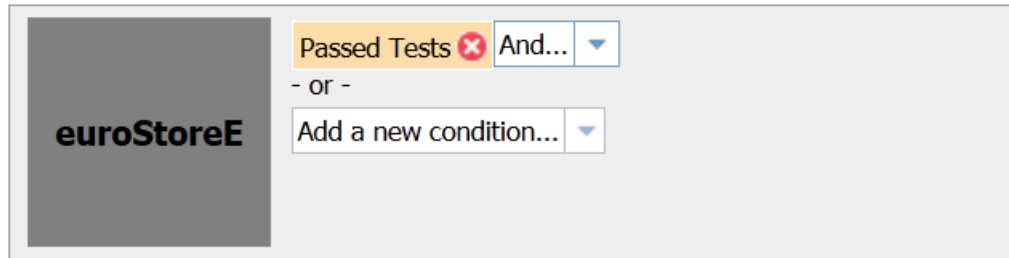
**Figure 45. Gates Pane**



2. Select a value from the Add a new condition list box.

The available statuses are defined in the default.xml file (discussed below). The default statuses—*Latest*, *Passed Tests*, *Archived*—are supplied as examples; it is assumed you will supply your own values.

Selecting a value provides both *And* and *Or* selection boxes.

**Figure 46. Gate Definition**

Using the *And* box adds an additional value to the condition that must be satisfied. Using the default values for example, defining the following gate *Passed Tests And Latest* means that only component versions with both statuses—*Passed Tests* and *Latest*—satisfy the condition and can be deployed into the environment. A single condition can have as many *And*-ed values as there are statuses defined in the `default.xml` file.

Using the *Or* box adds an additional condition to the gate. Additional conditions are defined in the same way as the first one. A gate with two or more conditions means the component will be deployed if it meets *any* of the conditions. For example, if the following two gates are defined, *Passed Tests*, and *Latest*, a component will pass the gate if it has either status (or both). A single gate can have any number of conditions.

3. Save your work when finished.

See the section called “Component Version Statuses” for more information about component statuses.

---

# Generic Processes

Generic processes are processes designed to run outside normal component or application processing. Generic processes can be used anywhere where an application or component process might not be needed. Generic processes are run by agents on hosts managed by the agents.

## Creating Generic Processes

Generic processes consist of plug-in steps like component processes, and are created in the same way component processes are created: steps are placed and configured in the process design editor. All plug-ins can be used but, obviously, some might be of little or no use outside a deployment. See the section called “Process Editor”.

## Running Generic Processes

To run a generic process, select the process and resource (the agent that will run the process), then use `Submit (Processes > [selected process])`.

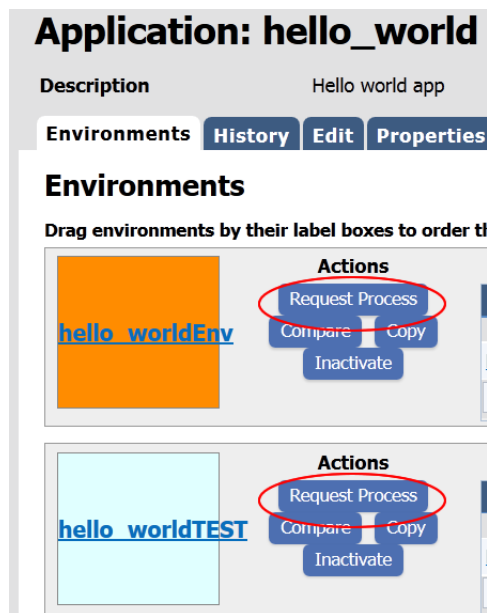
Generic processes can also be run from within component processes.

---

# Deployments

Deployments are done with applications (see the section called “Creating Applications” for information about creating applications). Performing a deployment is straightforward: you run a deployment-type process defined for an application in one of its environments. (Application processes can do things other than deploying, such as rolling-back or uninstalling components.) An application process is run by the Request Process command on the application's Environment pane (Application > *selected\_application* > Environment).

**Figure 47. Request Process Actions**



## To run an application:

1. In the IBM uDeploy web application, display the Application tab.
2. Click the name of the application.
3. Use the Request Process action for the environment where you want the deployment performed. The Run Process dialog is displayed.

In the illustration above, the application has two environments defined for it; you would click the Run Process link for the environment you want to use.

**Figure 48. Run Process Dialog**

Run Process on hello\_worldEnv

**Snapshot** None

**Version for hello\_world** None

**Only Changed Versions** ☒

**Process \*** hello\_worldAppProc (copy)

**Schedule Deployment?** ☐

Submit Cancel

4. If you want to use a snapshot, select it from the Snapshot drop-down list-box. If you select a snapshot, the deployment will automatically use the component version(s) defined for the snapshot. For information about snapshots, see the section called “Snapshots”.
5. If you did not select a snapshot, select a component version from the Version list-box. If more than one component is mapped to the application, each one is listed separately. Version options are described in the following table:

**Table 23. Version Options**

Version Option	Description
<b>None</b>	No version for this component. Useful when performing multi-component deployments or testing.
<b>Specific Version</b>	Enables you select any version already in Codestation.
<b>Latest Version</b>	Automatically uses the most recently imported version.
<b>Latest With Status</b>	All versions (creation order—oldest to newest) with the selected status. This might effect multiple versions, which is useful for an incremental component. Default values are: Latest, Passed Test, Archived.
<b>All With Status</b>	All component versions with the selected status will be deployed. Default values are: Latest, Passed Test, Archived.
<b>All in Environment</b>	Deploys all versions (in order of deployment) with the given <i>inventory</i> status in the current environment; useful if you need to run an operational process for whatever is already in the environment. Default values are: Active, Staged.

Version Option	Description
<b>All in Environment (Reversed)</b>	All component versions already deployed in the environment with the selected inventory status will be deployed in reverse order.

- Use the Only Changed Version check box to ensure that only changed versions are deployed (it is checked by default). If checked, no previously deployed versions will be deployed. If, for example, you check the box and select a specific version that was already deployed, the version will *not* be redeployed. Uncheck the box if you want to deploy a version regardless of whether or not it was already deployed (if the inventory is out of date, for instance).
- Select the process you want to run from the Process list box. All processes created for the application are listed.
- If you want to run the process at a later time, click the Schedule Deployment? check box (it is unchecked by default). If checked, fields appear enabling you to specify the date and time when the process will run. You can also make the process run on a recurring basis.
- When finished, click Submit to start the process. An application process will start immediately unless scheduled for a later time.

When a process starts, use the Application Process Request pane to review the deployment's status. This pane is also used if the process requires approvals.

**Figure 49. Application Process Request Pane**

### Application Process Request: doubleH

**Process**  
**Environment**  
**Only Changed Versions**  
**Date Requested**  
**Requested By**  
**Scheduled For**

[wH \(Version 3\)](#)  
[wH](#)  
false  
5/25/12 1:54 PM  
admin  
5/25/12 1:54 PM  
[View Deployment Request](#)

Log Properties Manifest

#### Approval Progress

Task	Role	Target	Status	Completed By	Actions
Environment Approval	Admin	<a href="#">wH</a>	Complete	admin on 5/25/12 1:54 PM	

#### Execution

Expand All Collapse All
Sort By: [Graph Order](#) [Start Time](#)

Step	Progress	Start	Duration	Status	Actions
hw1	1 of 1	1:54:16 PM	0:00:03	Success	
<a href="#">remote-agent1</a>	1 of 1	1:54:16 PM	0:00:03	Success	
<a href="#">1.0</a>		1:54:16 PM	0:00:03	Success	<a href="#">Details</a>
hwV	0 of 0	1:54:16 PM	0:00:00	Success	
<b>Total Execution</b>	<b>1 of 1</b>	<b>1:54:16 PM</b>	<b>0:00:03</b>	<b>Success</b>	

After a process finishes, click the Details action to display the Deployment of Component pane, which can be used to review the deployment details.




**Figure 50. Deployment of Component Pane**

**Deployment of Component: hello\_world**

**Process** [hello\\_worldInstall \(Version 3\)](#)  
**Version** 1.0  
**Resource** [remote-agent1](#)  
**Date** 5/25/12 4:32 PM  
**Requested By** admin  
[View Application Process Execution](#)

Log **Properties**

Sort By: [Graph Order](#) [Start Time](#)

Step	Type	Start	Duration	Status	Actions
Download Artifacts	<a href="#">UrbanCode Versioned File Storage v. 10.249680</a>	4:32:29 PM	0:00:02	Success	  
<b>Total Execution</b>		<b>4:32:29 PM</b>	<b>0:00:02</b>	<b>Success</b>	

The actions available for this pane enable you to review the deployment's output log, error log, and input/output parameters.

## Scheduling Deployments

IBM uDeploy has a built-in deployment scheduling system for setting regular deployments, or even black-out dates, for your Deployments. Deployments for an individual Application are scheduled on a per-environment basis, set when you run a deployment of a Snapshot or Deployment Process. Black-out dates are set within the individual Environments.

### Creating a Schedule

To set up a Scheduled Deployment, go to Application > Environment > Run Process. If you are scheduling a Snapshot deployment, you would go to Application > Snapshots > Run Process instead. Regardless of the type of deployment you are scheduling, configuration is the same.

After you check the Schedule Deployment box, IBM uDeploy will prompt you to give the date and time you want the deployment to run. The Make Recurring setting will deploy the Application on a regular schedule. For example, if you are practicing Continuous Delivery, the Daily option will deploy the Application to the target Environment every day.

Once you have scheduled the deployment, it will be added to the Calendar. There, if you click on the Scheduled Deployment, you can edit, delete, or investigate the deployment.

### Setting Blackouts

A blackout is a set per-environment, per-application. Once set, no deployments (nor snapshots) can be scheduled to occur in that environment. Any previously scheduled deployments to the Environment will fail if they fall within the blackout date you set. To set up a blackout, go to (*Application > Environments > Calendar > Add Blackout*). If you need to set blackouts for more than one environment, you must do this for each individual one. IBM uDeploy will prompt you to give the dates and times for the blackout.



---

# Reports

IBM uDeploy provides deployment- and security-type reports:

- **Deployment reports** contain historical information about deployments. Data can be filtered in a variety of ways and reports can be printed and saved. In addition, you can save search criteria for later use. See the section called “Deployment Reports”
- **Security reports** provide information about user roles and privileges. See the section called “Security Reports”

For information about saving and printing reports, see the section called “Saving and Printing Reports”

The following tables summarize the out-of-the-box reports.

**Table 24. Deployment Reports**

Report	Description
Deployment Detail	Provides information about deployments executed during a user-specified reporting period. Each report row represents a deployment that executed during the reporting period and matched the filter conditions. See the section called “Deployment Detail Report”.
Deployment Average Duration	Average deployment times for applications executed during a user-specified reporting period. See the section called “Deployment Average Duration Report”.
Deployment Total Duration	Total deployment times for applications executed during a user-specified reporting period. See the section called “Deployment Total Duration Report”.
Deployment Count	Provides information about the number of deployments executed during a user-specified reporting period. See the section called “Deployment Count Report”.

**Table 25. Security Reports**

Report	Description
Application Security	Provides information about user roles and privileges defined for IBM uDeploy-managed applications. See the section called “Application Security Report”.
Component Security	Information about user roles and privileges defined for components. See the section called “Component Security Report”.
Environment Security	Information about user roles and privileges defined for environments. See the section called “Environment Security Report”.
Resource Security	Information about user roles and privileges defined for resources. See the section called “Resource Security Report”.

## Deployment Reports

Deployment Reports contain historical information about deployments, such as the total number executed and their average duration. Data can be filtered in a variety of ways and reports can be printed and saved. In addition, you can save search criteria for later use. See the section called “Saving and Printing Reports”

## Deployment Detail Report

The Deployment Detail Report provides information about deployments executed during a user-specified reporting period. Each report row represents a deployment that executed during the reporting period and matched the filter conditions.

Reports can be filtered in a variety of ways (discussed below), and columns selectively hidden. Reports can be saved and printed. See the section called “Saving and Printing Reports”.

When selected, the report runs automatically for the default reporting period--current month--and with all filters set to *Any*. The default report represents all deployments that ran during the current month.

## Deployment Detail Fields

Initially, all fields are displayed.

**Table 26. Deployment Detail Fields**

Field	Description
Application	Name of the application that executed the deployment.
Environment	Target environment of the deployment.
Date	Date and time when the deployment was executed.
User	Name of the user who performed the deployment.
Status	Final disposition of the deployment. Possible values are: <ul style="list-style-type: none"> <li>• <i>Success</i></li> <li>• <i>Failure</i></li> <li>• <i>Running</i></li> <li>• <i>Scheduled</i></li> <li>• <i>Approval Rejected</i></li> <li>• <i>Awaiting Approval</i></li> </ul>
Duration	Amount of time the deployment ran. For a successful deployment, the value represents the amount of time taken to complete successfully. If deployment failed to start, no value is given. If a deployment started but failed to complete, the value represents the amount of time it ran before it failed or was cancelled.
Action	This field provides links to additional information about the deployment. The <i>View Request</i> link displays the <b>Application Process Request</b> pane. See <i>Applications</i> .

## Running the Deployment Detail Report

To run a report:

1. Use the *Date Range* date-picker to set the report's start- and end-dates.

**Table 27. Date Range**

Field	Description
Current, Prior Week	Start day is either Sunday or Monday, depending on what is defined in your system.
Current, Prior Month	Start day is first day of the month.
Current, Prior Quarter	Quarters are bound by calendar year.
Current, Prior Year	Current year includes today's date.
Custom	Displays the <b>Custom</b> pop-up which enables you to pick an arbitrary start- and end-date.

The start-time is automatically set to 00:00 (24-hour clock) for the selected date in the reporting period.

The end-time is automatically set to 24:00 for the selected date.

2. Set the report filters.

Filters are set with the properties drop-down list boxes. To set a filter, select it from the corresponding property list box.

**Table 28. Report Filters**

Field	Description
Application	Only deployments executed by the selected application appear in the report. Default value: <i>Any</i> .
Environment	Only deployments executed by the application selected with the <b>Application</b> list box that also used this environment appear in the report. If the application value is <i>Any</i> , the available value is <i>Any</i> ; otherwise, environments defined for the selected application are listed.
User	Only deployments executed by the selected user appear in the report. Default value: <i>Any</i> .
Status	Only deployments with the selected status appear in the report. Default value: <i>Any</i> .
Plugin	Only deployments that used the selected plug-in appear in the report. Default value: <i>Any</i> . Note: the <i>Any</i> value also includes deployments that did not use a plug-in.

3. Run the report.

Click the **Run** button to apply your filter conditions to the data and produce the report.

By default, the report is sorted by *Application*. You can sort the report on any field by clicking on the column header.

For information about saving and printing reports, see the section called “Saving and Printing Reports”.

## Sample Reports

The following table contains examples of reports that can be produced using the Deployment Detail Report.

**Table 29. Sample Reports**

Field	Description
<b>Show me:</b> <i>All failed deployments that occurred on July 4th during the previous year.</i>	<ul style="list-style-type: none"> <li>• Application: Any</li> <li>• Status: Failure</li> <li>• Date Range: Use the Custom pop-up to set the start- and end-dates to July 4th.</li> </ul>
<b>Show me:</b> <i>Deployments for an application that used a specific environment.</i>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select the value from the drop-down list box.</li> <li>• <b>Environment:</b> Select the environment from the drop-down list box.</li> </ul> <p>When an application is selected, only environments defined for it are available in the <b>Environment</b> drop-down list box.</p>
<b>Show me:</b> <i>Failed deployments that used a specific plug-in yesterday.</i>	<ul style="list-style-type: none"> <li>• <b>Status:</b> Failure</li> <li>• <b>Plugin :</b> Select the value from the drop-down list box.</li> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to set the start- and end-dates to the previous day.</li> </ul>
<b>Show me:</b> <i>My deployments that used a specific application during the past month.</i>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select the value from the drop-down list box.</li> <li>• <b>User:</b> Select your user ID.</li> <li>• <b>Date Range:</b> Select <i>Prior Month</i>.</li> </ul>

## Deployment Count Report

The Deployment Count Report provides information about the number of deployments executed during a user-specified reporting period. The report provides both a tabular presentation and line graph of the data. Each table row represents an environment used by an applications for the reporting period and interval.

The line graph elements are:

- y-axis represents the number of deployments
- x-axis represents reporting intervals
- plot lines represent environments used by applications

The units along the y-axis are scaled to the number of records reported. The units along the x-axis represent the reporting interval, which can be: months, weeks, or days. Each color-coded plot line represents a single environment used by the deployment during the reporting period.

When selected, the report runs automatically for the default reporting period (current month) and reporting interval (days), and with all filters set to *Any*. The default report provides a count of all deployments that ran during the current month.

## Deployment Count Table Fields

**Table 30. Deployment Count Fields**

Field	Description
Application	Name of the application that executed the deployment.
Environment	Name of the environment used by the application.
Reporting Interval	The remaining columns display the number of the deployments for the selected reporting interval.

## Running the Deployment Detail Report

To run a report:

1. Set the reporting period.

Use the *Date Range* date-picker to set the report's start- and end-dates. The selected value(s) determines the columns in the tabular report, and the coordinate interval on the graph's x-axis. Default value: *Current Month*.

**Table 31. Date Range**

Field	Description
Current, Prior Week	Start day is either Sunday or Monday, depending on what is defined in your system. Reporting interval is set to days.
Current, Prior Month	Start day is first day of the month. Reporting interval is set to days.
Current, Prior Quarter	Quarters are bound by calendar year. Reporting interval is set to weeks.
Current, Prior Year	Reporting interval is set months.
Custom	Displays the <b>Custom</b> pop-up which enables you to pick an arbitrary start- and end-date.

The start-time is automatically set to 00:00 (24-hour clock) for the selected date in the reporting period.

The end-time is automatically set to 24:00 for the selected date.

2. Set the report filters.

Filters are set with the properties drop-down list boxes. To set a filter, select it from the corresponding property list box.

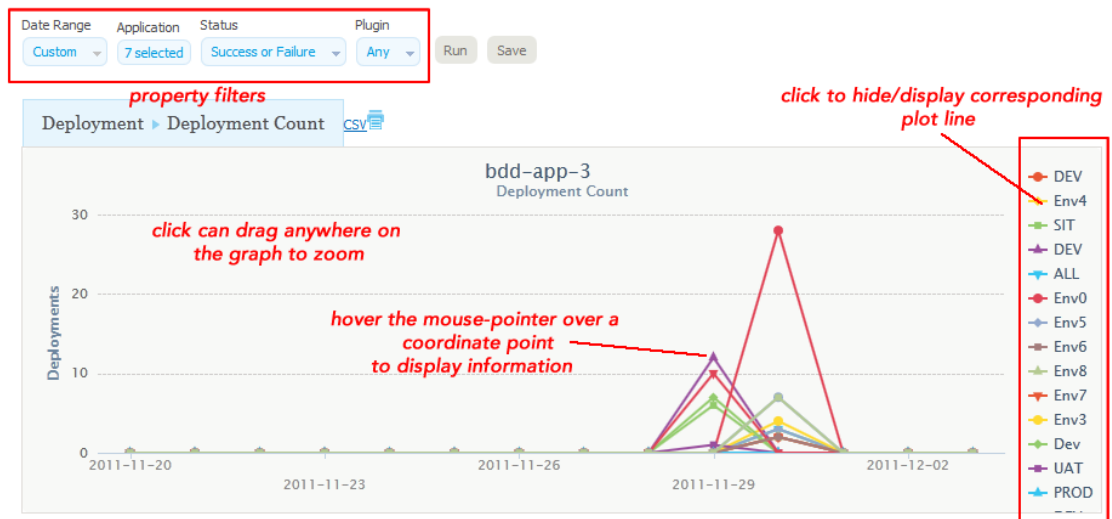
**Table 32. Filters**

Field	Description
Application	<p>Only deployments executed by the selected application(s) appear in the report. To select applications:</p> <ol style="list-style-type: none"> <li>Click the <b>Application</b> button.</li> <li>To include an application in the report, click the corresponding check box. If a large number of applications are listed, type the first few letters of the application's name in the text box to scroll the list. Multiple applications can be selected.</li> <li>Click <b>OK</b>.</li> </ol>
Status	Only deployments with the selected status appear in the report. Default value: <i>Success or Failure</i> , which means all deployments.
Plugin	Only deployments that used the selected plug-in appear in the report. Default value: <i>Any</i> . Note: the <i>Any</i> value also includes deployments that did not use a plug-in.

## 3. Run the report.

Click the **Run** button to apply your filter conditions to the data and produce the report.

A tabular report and line graph are produced.

**Figure 51. Deployment Count Graph**

Each environment used by a reporting application is represented by an individual plot line and table row. You can hide a plot line by clicking the corresponding item in the graph legend. To see information about a graph coordinate, hover the mouse over the graph point.

You can zoom a graph area by dragging the mouse over the area.

For information about saving and printing reports, see the section called “Saving and Printing Reports”.

## Sample Reports

The following table contains examples of reports that can be produced using the Deployment Count Report.

**Table 33. Sample Reports**

Field	Description
<b>Show me:</b> <i>The number of successful deployments for two specific applications during the past ten days that used a particular plug-in.</i>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select both applications from the <b>Applications</b> dialog.</li> <li>• <b>Status:</b> <i>Success</i></li> <li>• <b>Plugin:</b> Select the plug-in from the drop-down list box.</li> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to set the ten-day range.</li> </ul>
<b>Show me:</b> <i>The number of failed deployments for a given application during the past month</i>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select the value from the <b>Applications</b> dialog.</li> <li>• <b>Status:</b> <i>Failure</i></li> <li>• <b>Date Range:</b> Select <i>Prior Month</i>.</li> </ul>
<b>Show me:</b> <i>The number of failed deployments that used a specific plug-in yesterday.</i>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select the applications from the <b>Applications</b> dialog.</li> <li>• <b>Status:</b> <i>Failure</i></li> <li>• <b>Plugin:</b> Select the value from the drop-down list box.</li> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to select the previous day.</li> </ul>

## Deployment Average Duration Report

The Deployment Average Duration Report provides average deployment times for applications executed during a user-specified reporting period. The report provides both a tabular presentation and line graph of the data. Each table row represents an environment used by an application for the reporting period and interval.

The line graph elements are:

- y-axis represents deployment duration average times
- x-axis represents reporting intervals
- plot lines represent environments used by the applications

The units along the y-axis are scaled to the number of records reported. The units along the x-axis represent the reporting interval, which can be: months, weeks, or days. Each color-coded plot line represents a single environment used by the deployment during the reporting period.

When selected, the report runs automatically for the default reporting period (current month) and reporting interval (days), and with all filters set to *Any*. The default report provides average deployment times for all deployments that ran during the current month.

## Deployment Average Duration Fields

**Table 34. Average Duration Fields**

Field	Description
Application	Name of the application that executed the deployment.
Environment	Name of the environment used by the application.
Reporting Interval	The remaining columns display the average deployment times for the reporting interval.

## Running the Deployment Average Duration Report

To run a report:

1. Set the reporting period.

Use the *Date Range* date-picker to set the report's start- and end-dates. The selected value(s) determines the columns in the tabular report, and the coordinate interval on the graph's x-axis. Default value: *Current Month*.

**Table 35. Date Range**

Field	Description
Current, Prior Week	Start day is either Sunday or Monday, depending on what is defined in your system. Reporting interval is set to days.
Current, Prior Month	Start day is first day of the month. Reporting interval is set to days.
Current, Prior Quarter	Quarters are bound by calendar year. Reporting interval is set to weeks.
Current, Prior Year	Reporting interval is set months.
Custom	Displays the <b>Custom</b> pop-up which enables you to pick an arbitrary start- and end-date.

The start-time is automatically set to 00:00 (24-hour clock) for the selected date in the reporting period.

The end-time is automatically set to 24:00 for the selected date.

2. Set the report filters.

Filters are set with the properties drop-down list boxes. To set a filter, select it from the corresponding property list box.



**Table 36. Filters**

Field	Description
Application	<p>Only deployments executed by the selected application(s) appear in the report. To select applications:</p> <ol style="list-style-type: none"> <li>Click the <b>Application</b> button.</li> <li>To include an application in the report, click the corresponding check box.</li> </ol> <p>If a large number of applications are listed, type the first few letters of the application's name in the text box to scroll the list. Multiple applications can be selected.</p> <ol style="list-style-type: none"> <li>Click <b>OK</b>.</li> </ol>
Status	<p>Only deployments with the selected status appear in the report. Default value: <i>Success or Failure</i>, which means all deployments.</p>
Plugin	<p>Only deployments that used the selected plug-in appear in the report. Default value: <i>Any</i>. Note: the <i>Any</i> value also includes deployments that did not use a plug-in.</p>

3. Run the report.

Click the **Run** button to apply your filter conditions to the data and produce the report.

A tabular report and line graph are produced. Each environment used by a reporting application is represented by an individual plot line and table row. You can hide a plot line by clicking the corresponding item in the graph legend. To see information about a graph coordinate, hover the mouse over the graph point.

You can zoom a graph area by dragging the mouse over the area.

For information about saving and printing reports, see the section called “Saving and Printing Reports”.

## Sample Reports

The following table contains examples of reports that can be produced using the Deployment Average Duration Report.

**Table 37. Sample Reports**

Field	Description
<b>Show me:</b> <i>Average durations for two specific applications during the past ten days that used a particular plug-in.</i>	<ul style="list-style-type: none"> <li><b>Application:</b> Select both applications from the <b>Applications</b> dialog.</li> <li><b>Status:</b> <i>Success or Failure</i></li> <li><b>Plugin:</b> Select the plug-in from the drop-down list box.</li> </ul>

Field	Description
	<ul style="list-style-type: none"> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to set the ten-day range.</li> </ul>
<b>Show me:</b> <i>Average durations for successful deployments for a given application during the past six months.</i>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select the application from the <b>Applications</b> dialog.</li> <li>• <b>Status:</b> <i>Success</i></li> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to set the range to the previous six months.</li> </ul>

## Deployment Total Duration Report

The Deployment Total Duration Report provides total deployment times for applications executed during a user-specified reporting period. The report provides both a tabular presentation and line graph of the data. Each table row represents an environment used by one of the selected applications for the reporting period and interval.

The line graph elements are:

- y-axis represents deployment duration times
- x-axis represents reporting intervals
- plot lines represent environments used by the applications

The units along the y-axis are scaled to the number of records reported. The units along the x-axis represent the reporting interval, which can be: months, weeks, or days. Each color-coded plot line represents a single environment used by an application during the reporting period.

When selected, the report runs automatically for the default reporting period (current month) and reporting interval (days), and with all filters set to *Any*. The default report provides total deployment times for all deployments that ran during the current month.

## Deployment Total Duration Fields

**Table 38. Total Duration Fields**

Field	Description
Application	Name of the application that executed the deployment.
Environment	Name of the environment used by the application.
Reporting Interval	The remaining columns display the total deployment times for the reporting interval.

## Running the Deployment Total Duration Report

To run a report:

1. Set the reporting period.

Use the *Date Range* date-picker to set the report's start- and end-dates. The selected value(s) determines the columns in the tabular report, and the coordinate interval on the graph's x-axis. Default value: *Current Month*.

**Table 39. Date Range**

Field	Description
Current, Prior Week	Start day is either Sunday or Monday, depending on what is defined in your system. Reporting interval is set to days.
Current, Prior Month	Start day is first day of the month. Reporting interval is set to days.
Current, Prior Quarter	Quarters are bound by calendar year. Reporting interval is set to weeks.
Current, Prior Year	Reporting interval is set months.
Custom	Displays the <b>Custom</b> pop-up which enables you to pick an arbitrary start- and end-date.

The start-time is automatically set to 00:00 (24-hour clock) for the selected date in the reporting period.

The end-time is automatically set to 24:00 for the selected date.

2. Set the report filters.

Filters are set with the properties drop-down list boxes. To set a filter, select it from the corresponding property list box.

**Table 40. Filters**

Field	Description
Application	<p>Only deployments executed by the selected application(s) appear in the report. To select applications:</p> <ol style="list-style-type: none"> <li>Click the <b>Application</b> button.</li> <li>To include an application in the report, click the corresponding check box.</li> </ol> <p>If a large number of applications are listed, type the first few letters of the application's name in the text box to scroll the list. Multiple applications can be selected.</p> <ol style="list-style-type: none"> <li>Click <b>OK</b>.</li> </ol>
Status	Only deployments with the selected status appear in the report. Default value: <i>Success or Failure</i> , which means all deployments.
Plugin	Only deployments that used the selected plug-in appear in the report. Default value: <i>Any</i> . Note: the <i>Any</i> value also includes deployments that did not use a plug-in.

3. Run the report.

Click the **Run** button to apply your filter conditions to the data and produce the report.

A tabular report and line graph are produced. Each environment used by a reporting application is represented by an individual plot line and table row. You can hide a plot line by clicking the corresponding item in the graph legend. To see information about a graph coordinate, hover the mouse over the graph point.

You can zoom a graph area by dragging the mouse over the area.

For information about saving and printing reports, see the section called “Saving and Printing Reports”.

## Sample Reports

The following table contains examples of reports that can be produced using the Deployment Total Duration Report.

**Table 41. Sample**

Field	Description
<b>Show me:</b> <i>Total duration times for two specific applications during the past ten days that used a particular plug-in.</i>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select both applications from the <b>Applications</b> dialog.</li> <li>• <b>Status:</b> <i>Success or Failure</i></li> <li>• <b>Plugin:</b> Select the plug-in from the drop-down list box.</li> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to set the ten-day range.</li> </ul>
<b>Show me:</b> <i>Total duration times for successful deployments for a given application during the past six months.</i>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select the application from the <b>Applications</b> dialog.</li> <li>• <b>Status:</b> <i>Success</i></li> <li>• <b>Time Unit:</b> <i>Months</i></li> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to set the six-month range.</li> </ul>

## Security Reports

Security Reports provide information about user roles and privileges defined with the IBM uDeploy security system.

### Application Security Report

The Application Security Report provides information about user roles and privileges defined for IBM uDeploy-managed applications. Each report row represents an individual application. When selected, the report runs automatically for all applications.

## Application Security Fields

**Table 42. Application Security Fields**

Field	Description
Application	Name of the application.
Run Component Processes	Users who have component process execution privileges. For information about component processes, see the section called “Creating Components”.
Execute	Users who have application execution privileges. For information about applications, see <i>Applications</i> .
Security	Users who can define privileges for other users. For information about security, see ???.
Read	Users who can review information about the application but not change it.
Write	Users who can access and edit the application.

The report is sorted by *Application*. You can change the sort order by clicking on the column header.

For information about saving and printing reports, see the section called “Saving and Printing Reports”.

## Component Security Report

The Component Security Report provides information about user roles and privileges defined for components. Each report row represents an individual component. When selected, the report runs automatically for all components.

## Component Security Fields

Fields are:

**Table 43. Component Security Fields**

Field	Description
Component	Name of the component.
Execute	Users who have component process execution privileges. For information about component processes, see the section called “Creating Components”.
Security	Users who can define privileges for other users. For information about security, see ???
Read	Users who can review information about the component but not change it.
Write	Users who can access and edit the component.

The report is sorted by *Component*. You can change the sort order by clicking on the column header.

For information about saving and printing reports, see the section called “Saving and Printing Reports”.

## Environment Security Report

The Environment Security Report provides information about user roles and privileges defined for environments. Each report row represents an individual environment. When selected, the report runs automatically for all environments.

### Environment Security Fields

**Table 44. Environment Security Fields**

Field	Description
Application	Name of the application.
Environment	Name of the environment.
Execute	Users who have execution privileges for the environment. For information about environments, see <i>Applications</i> .
Security	Users who can define privileges for other users. For information about security, see ???.
Read	Users who can review information about the environment (but not change it).
Write	Users who can access and edit the environment.

The report can be sorted by *Application* or *Environment*. By default, it is sorted by *Application*. You can change the sort order by clicking on the column header.

For information about saving and printing reports, see the section called “Saving and Printing Reports”.

## Resource Security Report

The Resource Security Report provides information about user roles and privileges defined for resources. Each report row represents an individual resource. When selected, the report runs automatically for all resources.

### Resource Security Fields

Fields are:

**Table 45. Resource Security Fields**

Field	Description
Resource	Name of the resource.
Execute	Users who have execution privileges for the resource. For information about resources, see <i>Resources</i> .
Security	Users who can define privileges for other users. For information about security, see ???.
Read	Users who can review information about the resource but not change it.

Field	Description
Write	Users who can access and edit the resource.

The report is sorted by *Resource*. You can change the sort order by clicking on the column header.

For information about saving and printing reports, see the section called “Saving and Printing Reports”.

## Saving and Printing Reports

You can print and save report data for all report types. In addition, you can save filter and sort order information for deployment-type reports.

### Saving Report Data

IBM uDeploy saves report data in CSV files (comma separated value).

#### To save report data:

1. Set the filters (if any) and run the report.
2. Click the **CSV** button.
3. Use the **Opening File** dialog. You can save the data to file, or open the data with an application associated with CSV-type files on your system.

#### Note

Sort-order and hidden/visible column information is not preserved in the CSV file.

### Saving Report Filters

You can save filter and sort-order settings for deployment reports. Saved reports can be retrieved with the **My Reports** menu on the **Reports** pane.

#### To save a report:

1. Set the filter conditions.
2. Define the reporting period.
3. Run the report.
4. Optionally, set the sort order. You can change the sort order for any column by clicking the column header.
5. Optionally, change column visibility. Click the **Edit** button to display the Select Columns dialog. By default, all columns are selected to appear in a report. To hide a column, click the corresponding check box.
6. Click the **Save** button. The **Save Current Filters** dialog is displayed.
7. Enter a name for the file, and save your work.

To run your report, click the report name in the **My Reports** menu.

To delete your report, click the **Delete** button.

## Printing Reports

**To print a report:**

1. Set the filter conditions.
2. Define the reporting period.
3. Run the report.
4. Optionally, set the sort order. Your changes are reflected in the printed report.
5. Optionally, change column visibility. By default, all columns are selected to appear in the printed report. Hidden columns will not appear in the output.
6. Click the **Print** button to print your report.



---

# Reference

---

---

# Component Source Configuration

## Basic Fields

These fields appear for all source types; they are displayed when the Create New Component dialog opens. Other fields, discussed below, are displayed when a source type is selected.

**Table 46. Fields Available for All Source Types**

Field	Description
<b>Name</b>	Identifies the component; appears in many UI features. Required.
<b>Description</b>	The optional description can be used to convey additional information about the component. If the component is used by more than one application, for example, entering "Used in applications A and B" can help identify how the component is used.
<b>Template</b>	<p>A component template enables you to reuse component definitions; components based on templates inherit the template's source configuration, properties, and process. Any previously created templates are listed. A component can have a single template associated with it. The default value is <i>None</i>.</p> <p>If you select a template, the Template Version field is displayed which is used to select a template version. By controlling the version, you can roll-out template changes as required. The default value is Latest Version which means the component will automatically use the newest version (by creation date). See the section called “Component Templates”.</p> <p><b>Note</b></p> <p>If you select a template that has a source configured for it, the dialog box will change to reflect values defined for the template. Several fields, including the Source Config Type field, will become populated and locked.</p>
<b>Source Config Type</b>	Defines the source type for the component's artifacts; all artifacts must have the same source type. Selecting a value displays additional fields associated with the selection. Source-dependent fields (see <i>Component Source Configuration</i> ) are used to identify and configure the component's artifacts. If you selected a template, this field is locked and its value is inherited from the template.
<b>Import Versions Automatically</b>	If checked, the source location is periodically polled for new versions; any found are automatically imported. The default polling period is 15 seconds, which can be changed with the System Settings pane. If left unchecked, you can manually create versions by using the Versions pane. By default, the box is unchecked.
<b>Copy to CodeStation</b>	This option—selected by default—creates a tamper-proof copy of the artifacts and stores them in the embedded artifact management system, CodeStation. If unchecked, only meta data about the artifacts are imported. UrbanCode, an IBM Company recommends that the box be checked.

Field	Description
<b>Default Version Type</b>	Defines how versions are imported into CodeStation. <code>Full</code> means the version is comprehensive and contains all artifacts; <code>Incremental</code> means the version contains a subset of the component's artifacts. Default value is: <code>Full</code> . Required.
<b>Inherit Cleanup Settings</b>	Determines how many component versions are kept in CodeStation, and how long they are kept. If checked, the component will use the values specified on the System Settings pane. If unchecked, the Days to Keep Versions (initially set to -1, keep indefinitely) and Number of Versions to Keep (initially set to -1, keep all) fields are displayed, which enable you to define custom values. The default value is checked.

## File System (Basic and Versioned)

See the section called “Basic Fields” for information about the standard fields which apply to each source type.

### File System (Basic)

Imports everything in the target directory whenever you import versions. You can set up a template to auto-increment version numbers. Automatic import is not available for this source type.

**Table 47. File System (Basic) Source Fields**

Field	Description
<b>Base Path</b>	Defines how the property is presented to users in the IBM uDeploy editor. This element has several attributes:
<b>Always Use Name Pattern</b>	Used to specify values for a select-box. Each value has a mandatory <code>label</code> attribute which is displayed to users, and a value used by the property when selected. Values are displayed in the order they are defined.
<b>Version Name Pattern</b>	Defines how the property is presented to users in the IBM uDeploy editor. This element has several attributes:
<b>Next Version Number</b>	Defines how the property is presented to users in the IBM uDeploy editor. This element has several attributes:
<b>Save File Execute Bits</b>	Defines how the property is presented to users in the IBM uDeploy editor. This element has several attributes:

### File System (Versioned)

The File System (Versioned) source type interacts with file-system-based artifacts. It assumes that subdirectories within the base directory are distinct component versions. File System (Versioned) can automatically import versions into CodeStation.

**Table 48. File System (Versioned) Source Fields**

Field	Description
<b>Base Path</b>	Path to directory containing artifacts. The content of each subdirectory within the base directory is considered a distinct component version.

Field	Description
	The subdirectory with the most recent time-stamp is considered the "latest version."
Save File Execute Bits	Defines how the property is presented to users in the IBM uDeploy editor. This element has several attributes:

## Serena Dimensions CM

Serena Dimensions CM is a software configuration management tool. To use Serena Dimensions CM as an artifact source, select *Dimensions* from the Source Config Type drop-down list box then configure the type-specific fields described here. For information about creating components, see the section called “Creating Components”.

See the section called “Basic Fields” for information about the standard fields which apply to each source type.

**Table 49. Serena Dimensions Fields**

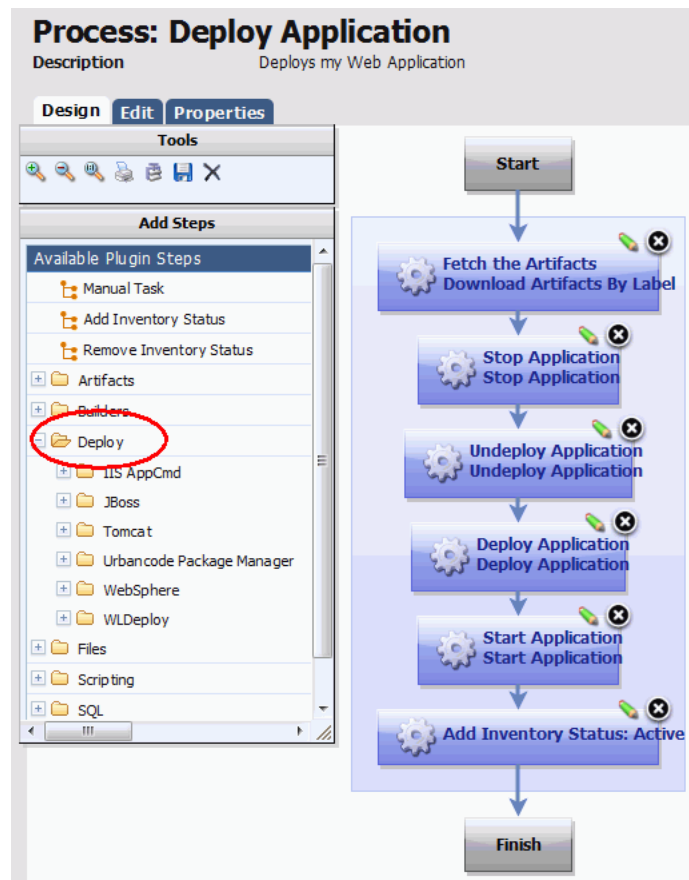
Field	Description
Username	Dimensions CM user name. For information about user impersonation, see ???.
Password	Password associated with the Dimensions user name.
DB Name	Name of the Dimensions database.
DB Connection	Name of the Dimensions connection to be used. A connection/session is required in order to send or receive commands to/from the database.
Server	Server managing the Dimensions database.
Product Spec	Location of the Dimension-managed artifacts.

# Plug-ins

IBM uDeploy plug-ins provide tools for creating component processes. Plug-ins consist of configurable *steps* which can be thought of as distinct pieces of automation. By combining steps in the IBM uDeploy editor, you can create fully-automated deployment processes. In addition to basic plug-ins, others integrate many third-party tools into IBM uDeploy, such as application servers and software configuration management products. For example, the Tomcat and WebSphere plug-ins--to name just two--provide steps that start and stop those servers, install and uninstall applications, as well as perform other tool-specific tasks. Finally, you can write your own plug-in (see the section called “Creating Plug-ins”).

A plug-in consists of a number of steps, which varies from plug-in to plug-in. Each step consists of a number of properties, a command that performs the function associated with the step, and post-processing instructions (typically used to ensure that expected results occurred). Step properties can serve a wide variety of purposes, from providing information required by the step's command, to supplying some or all of the actual command itself. When you create a process, you drag steps onto the editor's design area and define their properties as you go. Property values can be supplied when defining a component process or at run-time. The process flow is defined by drawing connections between steps. In the following illustration, you can see a series of plug-in steps and the connections between them. For information about creating component processes, see the section called “Component Processes”; for information about creating your own post-processing scripts, see ???.

**Figure 52. Example Process**



## Plug-ins at Run-time

Component processes are run by agents installed in the target environment. For a process to run successfully, the agent must have access to all resources, tools, and files required by the plug-in steps used in the process. When installing an agent, ensure that:

- The agent running the process has the necessary user permissions to execute commands and access any required resources. This typically entails granting permissions if an external tool is installed as a different user; installing the agent as a service; or impersonating the appropriate user (see ???).
- Any external tools required by plug-in steps are installed in the target environment.
- The required minimum version of any external tool is installed.

For information about installing agents, see the section called “Agent Installation”.

## Standard Plug-ins

IBM uDeploy also includes a standard set of automation steps that can be used to add additional automation to any process. These will typically be used for advanced processes or where there is no standard integration step available from one of the integrations.

### Shell

The Shell integration consists of a single step that you can include in any deployment process or other process. The most common use case opening and running a shell script on the target machine. If the step is used within a larger process, ensure that you set the order correctly. For example, if you have to run a shell script prior to executing another process, you will need to add the Shell step above the other step.

### UrbanCode, an IBM Company Package Manager

This is for advanced usage. The steps work in conjunction with IBM uDeploy to create and manage application packages for deployments. These steps will not generally be used as part of a regular deployment.

### IBM uDeploy

These advanced automation steps will retrieve properties and environments from IBM uDeploy.

## Creating Plug-ins

A plug-in consists of two mandatory XML files--plugin.xml and upgrade.xml--along with any supporting script files required by the plug-in. The plugin.xml file defines the steps comprising the plug-in; a plug-in's functionality is defined by its steps. Each step is an independently configurable entity in the IBM uDeploy editor.

The upgrade.xml file is used to upgrade the plug-in to a new version. Optionally, you can include an info.xml file which contains a version ID and other information used by the UrbanCode, an IBM Company plug-in page. Although optional, UrbanCode, an IBM Company recommends the use of the info.xml file.

A plug-in step is defined by a <step-type> element that contains: one <properties> element, one <command> element, and one <post-processing> element. The <properties> element is a container for <property> child elements, and can contain any number of <property> elements. Property values can be supplied at design- or run-time. The <post-processing> element provides error-handling capabilities and sets property values that can be used by other steps. The <command> element performs the step's function. The function can be defined completely by the element, or be constructed in part or entirely from the step's properties at design- or run-time.

In addition to a step's own properties, a command has access to properties set earlier by other steps within the process, to properties set by the application that invoked the component process, as well as to those on the target environment and resource. Step property values become unavailable once the component process ends.

Plug-in steps are performed by an agent installed in the target environment, which means that plug-ins can be written in any scripting language as long as the agent can access the required scripting tools on the host. Once a plug-in is created, upload it into IBM uDeploy to make it available to users. To upload a plug-in, create a ZIP archive that contains the XML files (plugin.xml and upgrade.xml) along with any scripts required by the plug-in, then import the ZIP file with the Automation Plugins pane (Settings > Automation Plugins > Load Plugin).

## The plugin.xml File

A plug-in is defined with the plugin.xml file. The structure of this file consists of a header element and one or more step-type elements. The header identifies the plug-in. Each step-type element defines a step; steps are available to users in the IBM uDeploy process editor and used to construct component processes.

After the document type declaration, the plugin root element identifies the XML schema type, PluginXMLSchema\_v1.xsd, which is used by all plug-ins. The following presents the basic structure of plugin.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin xmlns="http://www.UrbanCode, an IBM Company.com/PluginXMLSchema_v1"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <header>
    <identifier id="plugin_id" version="version_number" name="Plug-in Name"/>
    <description/>
    <tag>Plugin_type/Plugin_subtype/Plugin_name</tag>
  </header>
  <step-type name="Step_Name">
    <description/>
    <properties>
      <property name="property_name" required="true">
        <property-ui type="textBox" label="Driver Jar"
          description="The full path to the jdbc driver jar to use."
          default-value="{p:resource/sqlJdbc/jdbcJar}"/>
      </property>
    </properties>
    <post-processing>
      <![CDATA[
        if (properties.get("exitCode") != 0) {
          properties.put("Status", "Failure");
        }
        else {
          properties.put("Status", "Success");
        }
      ]]>
    </post-processing>

    <command program="{path_to_tool}">
```

```

    <arg value="parameters_passed_to_tool"/>
    <arg path="{p:jdbcJar}"/>
    <arg file="command_to_run"/>
    <arg file="{PLUGIN_INPUT_PROPS}"/>
    <arg file="{PLUGIN_OUTPUT_PROPS}"/>
  </command>
</step-type>
</plugin>

```

## The <header> Element

The mandatory `header` element identifies the plug-in and contains three child elements:

**Table 50.**

<header> Child Elements	Description
<identifier>	<p>This element's three attributes identify the plug-in:</p> <ul style="list-style-type: none"> <li>• <i>version</i></li> </ul> <p>API version (the version number used for upgrading plug-ins is defined in the <code>info.xml</code> file).</p> <ul style="list-style-type: none"> <li>• <i>id</i></li> </ul> <p>Identifies the plug-in.</p> <ul style="list-style-type: none"> <li>• <i>name</i></li> </ul> <p>The plug-in name appears on IBM uDeploy's web application Automation Plugins pane, and on the UrbanCode, an IBM Company.com plug-in page.</p> <p>All values must be enclosed within single-quotes.</p>
<description>	Describes the plug-in; appears on IBM uDeploy's web application Automation Plugins pane, and on the UrbanCode, an IBM Company.com plug-in page.
<tag>	Defines where the plug-in is listed within the IBM uDeploy editor's hierarchy of available plug-ins. The location is defined by a string separated by slashes. For example, the Tomcat definition is: <code>Application Server/Java/Tomcat</code> . The Tomcat steps will be listed beneath the Tomcat item, which in turn is nested within the other two.

The following is a sample header definition:

```

<header>
  <identifier version="3" id="com.&company;.air.plugin.Tomcat" name="Tomcat"/>
  <description>
    The Tomcat plugin is used during deployments to execute Tomcat run-book

```



```

    automations and deploy or undeploy Tomcat applications.
  </description>
  <tag>Application Server/Java/Tomcat</tag>
</header>

```

## Plug-in Steps--the <step-type> Element

Plug-in steps are defined with the `step-type` element; each `step-type` represents a single step in the IBM uDeploy process editor. A `step-type` element has a name attribute and several child elements: `description`, `properties`, `command`, and `post-processing`.

The mandatory name attribute identifies the step. The description and name appear in IBM uDeploy's web application and on the UrbanCode, an IBM Company.com plug-in page.

```

<step-type name="Start">
  <description>Start Apache HTTP server</description>

```

### Note

A step name cannot contain the "/" character.

## Step Properties--the <properties> Element

The `properties` element is a container for properties which are defined with the `property` tag. Each step has a single `properties` element; a `properties` element can contain any number of `property` child elements.

A `property` tag has a mandatory name attribute, optional `required` attribute, and two child elements, `property-ui` and `value`, which are defined in the following table.

**Table 51. The <property> Element**

<property> Child Elements	Description
<property-ui>	<p>Defines how the property is presented to users in the IBM uDeploy editor. This element has several attributes:</p> <ul style="list-style-type: none"> <li> <i>label</i>            Identifies the property in the editor's Edit Properties dialog box.         </li> <li> <i>description</i>            Text displayed to users in the associated roll-over help box.         </li> <li> <i>default-value</i>            Property value displayed when the Edit Properties dialog box is displayed; used if unchanged.         </li> <li> <i>type</i>            Identifies the type of widget displayed to users. Possible values are:         </li> </ul>

<property> Child Elements	Description
	<ul style="list-style-type: none"> <li>• <i>textBox</i>  Enables users to enter an arbitrary amount of text, limited to 4064 characters.</li> <li>• <i>textAreaBox</i>  Enables users to enter an arbitrary amount of text (larger input area than <i>textBox</i>), limited to limited to 4064 characters.</li> <li>• <i>secureBox</i>  Used for passwords. Similar to <i>textBox</i> except values are redacted.</li> <li>• <i>checkBox</i>  Displays a check box. If checked, a value of <i>true</i> will be used; otherwise the property is not set.</li> <li>• <i>selectBox</i>  Requires a list of one or more values which will be displayed in a drop-down list box. Configuring a value is described below.</li> </ul>
<value>	Used to specify values for a <i>selectBox</i> . Each value has a mandatory <i>label</i> attribute which is displayed to users, and a value used by the property when selected. Values are displayed in the order they are defined.

Here is a sample <property> definition:

```
<property name="onerror" required="true">
  <property-ui type="selectBox"
    default-value="abort"
    description="Action to perform when statement fails: continue, stop, abort."
    label="Error Handling"/>
  <value label="Abort">abort</value>
  <value label="Continue">continue</value>
  <value label="Stop">stop</value>
</property>
```

## The <command> Element

Steps are executed by invoking the scripting tool or interpreter specified by the <command> element. The <command> element's *program* attribute defines the location of the tool that will perform the command. It bears repeating that the tool must be located on the host and the agent invoking the tool must have access to it. In the following example, the location of the tool that will perform the command--the Java-based scripting tool *groovy* in this instance--is defined.

```
<command program='${GROOVY_HOME}/bin/groovy'>
```

The actual command and any parameters it requires are passed to the tool by the `<command>` element's `<arg>` child element. Any number of `<arg>` elements can be used. The `<arg>` element has several attributes:

**Table 52. `<arg>` Element Attributes**

Attribute	Description
<code>&lt;value&gt;</code>	Specifies a parameter passed to the tool. Format is tool-specific; must be enclosed by quotes.
<code>&lt;path&gt;</code>	Path to files or classes required by the tool. Must be enclosed by quotes.
<code>&lt;file&gt;</code>	Specifies the path to any files or classes required by the tool. Format is tool-specific; must be enclosed by quotes.

Because `<arg>` elements are processed in the order they are defined, ensure the order conforms to that expected by the tool.

```

<command program='${GROOVY_HOME}/bin/groovy'>
  <arg value='-cp' />
  <arg path='classes:${sdkJar}:lib/commons-codec.jar:
    lib/activation-1.1.1.jar:
    lib/commons-logging.jar:lib/httpclient-cache.jar:
    lib/httpclient.jar:lib/httpcore.jar:
    lib/httpmime.jar:lib/javamail-1.4.1.jar' />
  <arg file='registerInstancesWithLB.groovy' />
  <arg file='${PLUGIN_INPUT_PROPS}' />
  <arg file='${PLUGIN_OUTPUT_PROPS}' />
</command>

```

The `<arg file='${PLUGIN_INPUT_PROPS}' />` specifies the location of the tool-supplied properties file. The `<arg file='${PLUGIN_OUTPUT_PROPS}' />` specifies the location of the file that will contain the step-generated properties.

Note: new lines are not supported by the `<arg>` element and are shown in this example only for presentation.

## The `<post-processing>` Element

When a plug-in step's `<command>` element finishes processing, the step's mandatory `<post-processing>` element is executed. The `<post-processing>` element sets the step's output properties (step name/property name, see *IBM uDeploy Properties*) and provides error handling. The `<post-processing>` element can contain any valid JavaScript script (unlike the `<command>` element, `<post-processing>` scripts must be written in JavaScript). Users can also provide their own scripts when defining the step in the IBM uDeploy editor, see ???. Although not required, it's recommended that scripts be wrapped in a CDATA element.

You have access to a `java.util.Properties` variable called `properties`. The `properties` variable has several special properties: `exitCode` contains the process exit code, and `Status` contains the step's status. A `Status` value of `Success` means the step completed successfully.

Another available variable—`scanner`— can scan the step's output log (scanning occurs on the agent) and take actions depending on the results. `scanner` has several public methods:

- `register(String regex, function call)` registers a function to be called when the regular expression is matched.
- `addLOI(Integer lineNumber)` adds a line to the lines of interest list, which are highlighted in the Log Viewer; implicitly called whenever `scanner` matches a line.
- `getLinesOfInterest()` returns a `java.util.List` of lines of interest; can be used to remove lines.
- `scan()` scans the log. Use after all regular expressions are registered.

The post-processing script can examine the step's output log, and take actions based on the result. In the following code fragment, `scanner.register()` registers a string with a regular expression engine, then takes an action if the string is found. Once all strings are registered, it calls `scanner.scan()` on the step's output log line by line.

```
<![CDATA[
    properties.put("Status", "Success");
    if (properties.get("exitCode") != 0) {
        properties.put("Status", "Failure");
    }
    else {
        scanner.register("(?i)ERROR at line", function(lineNumber, line) {
            var errors = properties.get("Error");
            if (errors == null) {
                errors = new java.util.ArrayList();
            }
            errors.add(line);
            properties.put("Error", errors);

            properties.put("Status", "Failure");
        });
        .
        .
        .
        scanner.scan();

        var errors = properties.get("Error");
        if (errors == null) {
            errors = new java.util.ArrayList();
        }
        properties.put("Error", errors.toString());
    }
}]
```

You can use a post-processing scripts to set output properties that can be used in other steps in the same process, which enables complex workflows. Reference prior step output properties this way:

```
${p:stepName/propName}
```

## Upgrading Plug-ins

To create an upgrade, first, increment the number of the version attribute of the <identifier> element in plugin.xml. Next, create a <migrate> element in upgrade.xml with a to-version attribute containing the new number. Finally, place the property and step-type elements that match the updated plugin.xml file within this element, as shown in the following example.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin-upgrade
  xmlns="http://www.&company;.com/UpgradeXMLSchema_v1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <migrate to-version="3">
    <migrate-command name="Run SQLPlus script">
      <migrate-properties>
        <migrate-property name="sqlFiles" old="sqlFile"/>
      </migrate-properties>
    </migrate-command>
  </migrate>
  <migrate to-version="4">
    <migrate-command name="Run SQLPlus script" />
  </migrate>
  <migrate to-version="5">
    <migrate-command name="Run SQLPlus script" />
  </migrate>
</plugin-upgrade>
```

Of course, you can also make a script-only upgrade, that is, an upgrade that contains changes to the step's associated scripts and files but does not change plugin.xml. This mechanism can be useful for plug-in development and for minor bug-fixes/updates.

## The info.xml File

Use the optional info.xml file to describe the plug-in and provide release notes to users. The file's <release-version> element can be used for version releases.

## Example Plug-in

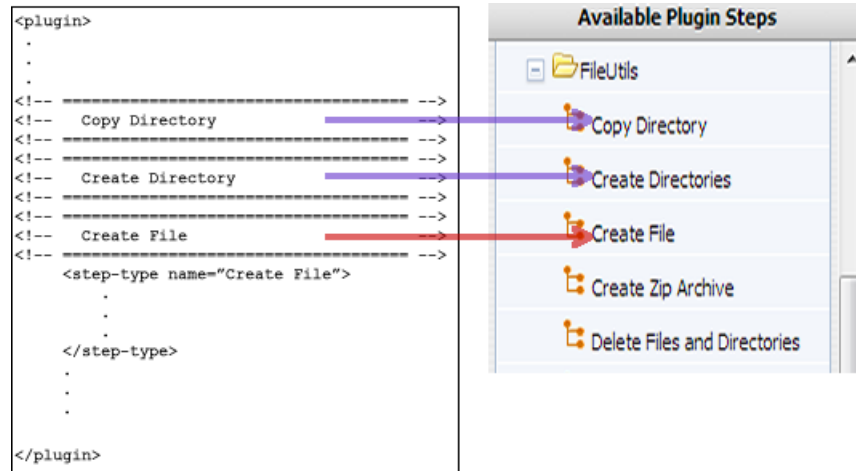
A plug-in consists of one or more steps. A step represents a unit of functionality that can be user-configured and combined with other steps into a process. Creating a plug-in consists in defining its individual steps and then grouping them together for presentation in IBM uDeploy.

Plug-in steps are performed by an agent installed in the target environment. What this means is that plug-ins can be written in any scripting language as long as the agent can access the required scripting tools on the host.

In this section, we examine the mechanics of plug-ins by examining a plug-in step in detail. The example plug-in we use is the UrbanCode, an IBM Company-created plug-in *FileUtils*, which contains several steps related to file manipulation. The FileUtils plug-in is shipped with IBM uDeploy.

Each plug-in step is an individually configurable object in the editor. In the following illustration you can see some of FileUtils' individual steps in the process editor.

**Figure 53. Plug-in Steps**



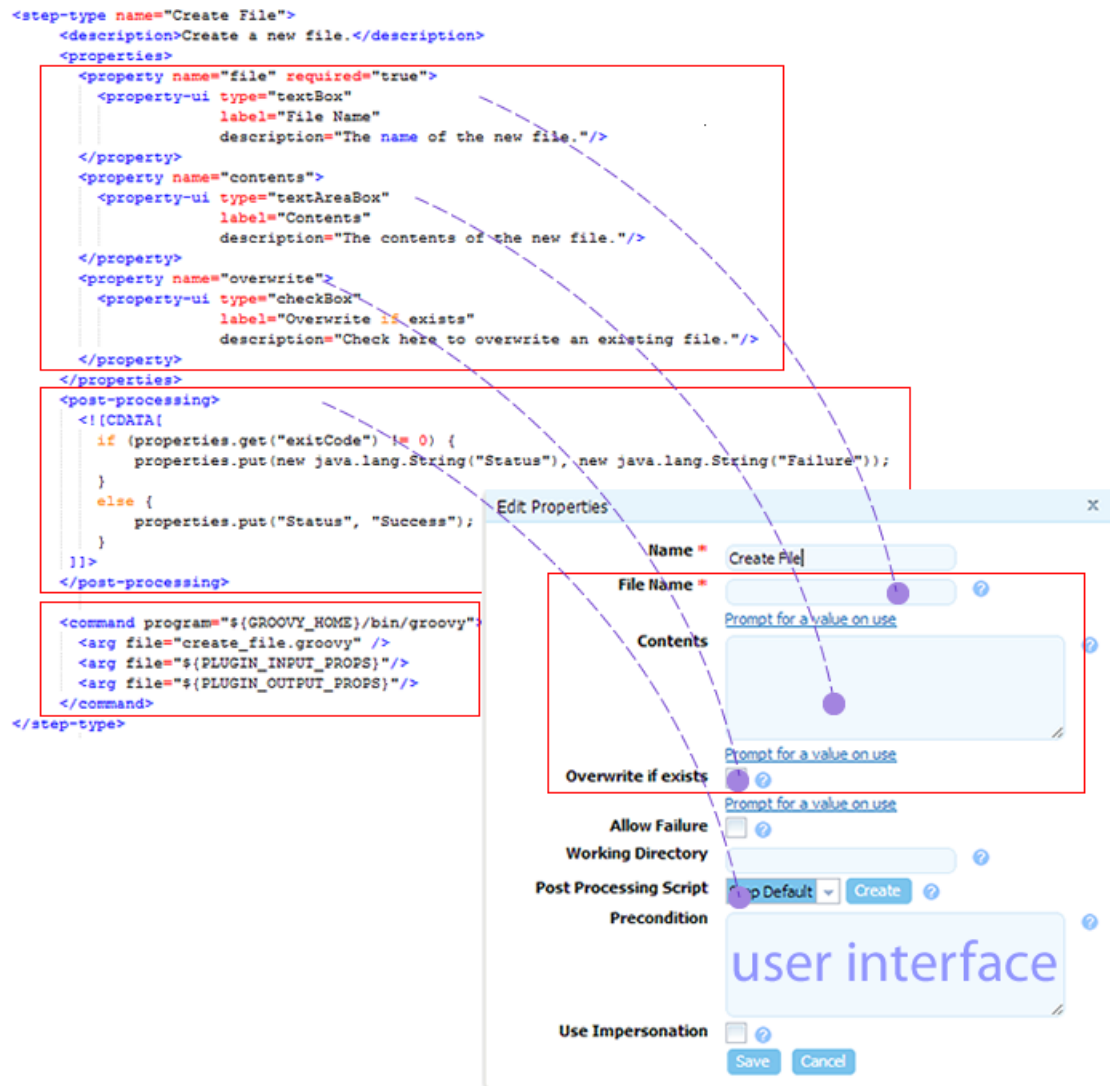
We examine the *Create File* step in this example. Create File is straightforward and—as the name implies—creates a file.

Each step—`step-type` element—has the same structure:

- `properties` element can contain any number of `<property>` child elements; property values can be supplied at design- or run-time
- `post-processing` element provides error-handling and sets property values that can be used by other steps
- `command` element performs the step's function; the function can be defined completely by the element, or be constructed in part or entirely from the step's properties at design- or run-time

The following figure illustrates the structure of the Create File step.

Figure 54. Create File Step Structure



## Step Properties

In the context of our discussion, properties are values that are used by the step's command. Step properties are defined with the `property` element.

As you can see in Figure 54, “Create File Step Structure”, the Create File step has three properties:

- *file* contains the name of the file the command will create; it is represented as a text box in the process designer
- *contents* will contain the file's content; it is represented by a text-area box, which can contain a large amount of data
- *overwrite* is used to specify whether the file can overwrite an existing file; it is represented by a check box in the process designer

These properties are displayed in the dialog box that appears when the step is added to a component process. The other properties in the dialog are displayed for every step (post-processing is discussed

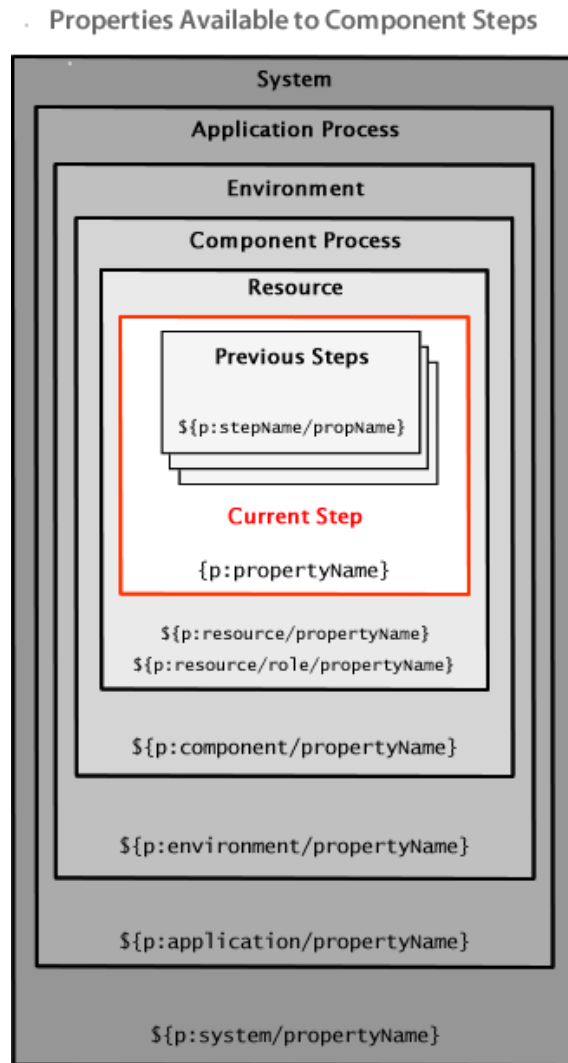
below). Property values can be entered into the dialog box by the process designer at design-time, or left to be furnished at run-time by the user running the application.

You can configure most properties with the `property-ui` child element (the `selectBox` type requires the `value` child element as well). See the section called “Step Properties--the `<properties>` Element” for information about the options available for presentation in the UI. Default values can be defined when you create the step.

In addition to a step's own properties, a command has access to properties set earlier by other steps within the process, to properties set by the application that invoked the component process, as well as to those on the target environment and resource.

The following illustrates the properties available to an individual step. Step property values become unavailable once the component process ends.

**Figure 55. Properties**



Run-time defined properties are combined with those defined earlier and together sent to the agent. Earlier properties—those defined outside the current process—are retrieved from the database. How properties are processed and consumed is discussed further in the next section.

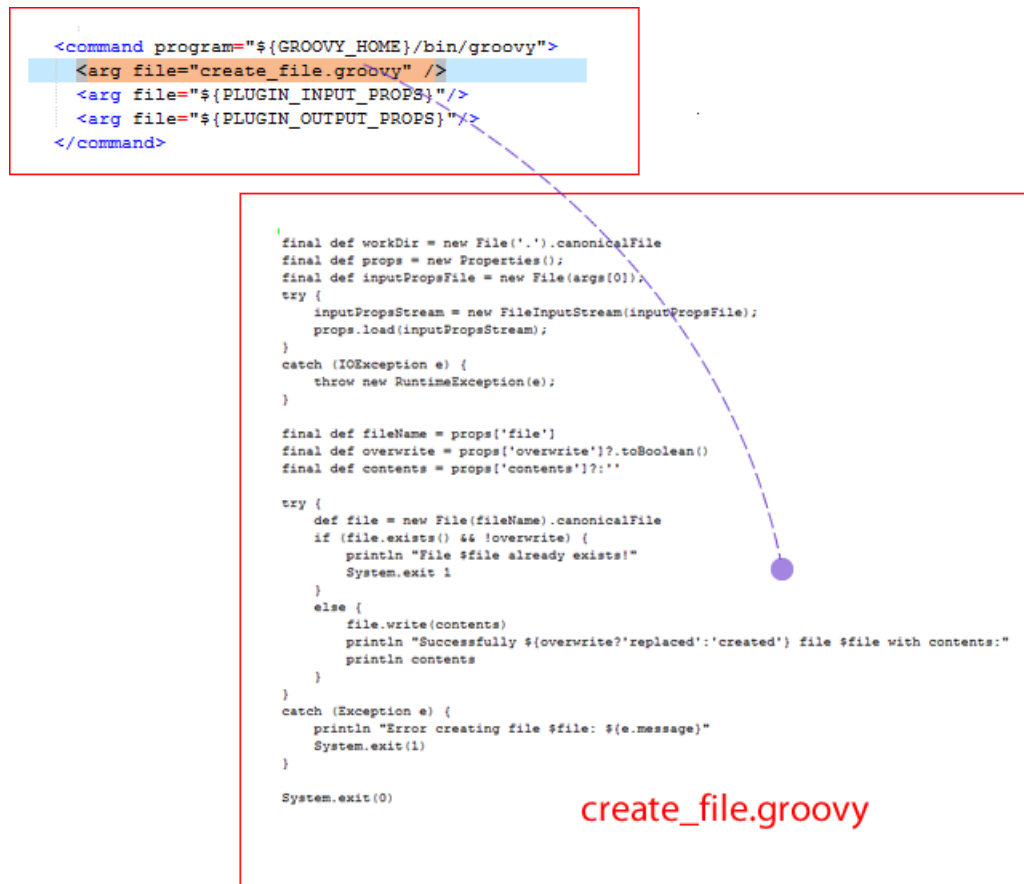


## Step Commands

The step's *command* element specifies the scripting tool that will perform the step and identifies the file containing the actual script. The agent that will perform the step will download the plug-in and expect to find the script among the downloaded files. Of course, the agent must also have access to the scripting tool. Any other arguments required by the script/tool can also be specified with the *arg file* attribute. The *arg file* attributes should be defined in the order compatible with the scripting tool.

The Create File command script is written in Groovy. Groovy is a Java-like scripting language.

**Figure 56.** create\_file.groovy



This command instructs the Groovy interpreter to run `create_file.groovy` (more about Groovy and the particulars of this file in the following section).

This line (which is part of every command):

```
<arg file="${PLUGIN_INPUT_PROPS}" />
```

sends a file containing the properties required by the step to the agent. The properties in the file are those furnished at run-time and others defined earlier that are required by the step. See Figure 55, “Properties”. The `${PLUGIN_INPUT_PROPS}` variable resolves to the location of this IBM uDeploy-managed properties file.

And this line (which is also part of every command):

```
<arg file="\${PLUGIN_OUTPUT_PROPS}"/>
```

refers to the file returned by the agent after finishing the step. The properties in this file are available to later steps in the process. The `\${PLUGIN_OUTPUT_PROPS}` variable resolves to the location of this IBM uDeploy-managed properties file.

## create\_file.groovy

The `create_file.groovy` file contains the Groovy script that will perform the step's command. Groovy is a dynamic scripting language (similar to Python, Ruby, and Perl) for the Java platform. Most Java code is also syntactically valid Groovy, which makes Groovy popular with Java programmers. Groovy provides native support for regular expressions.

This first lines of the script create a properties object, `props`, then attempts to load the properties from the file sent from the server (specified by the `\${PLUGIN_OUTPUT_PROPS}` variable). If it can load the file, it populates `props`; otherwise, it throws an exception.

```
final def workDir = new File('.').canonicalFile
final def props = new Properties();
final def inputPropsFile = new File(args[0]);
try {
    inputPropsStream = new FileInputStream(inputPropsFile);
    props.load(inputPropsStream);
}
catch (IOException e) {
    throw new RuntimeException(e);
}
```

To perform the command—create a file—the script uses the properties defined by the step itself. The script retrieves the three properties from `props` and creates corresponding local variables.

Next, the script creates a file with a name specified by `fileName`, and tests the `overwrite` boolean variable. If a file with the same name exists and `overwrite` is false, the script ends (fails) with an exit code of 1. Exit codes can be examined during post-processing.

Otherwise, the file is written with the content of `contents`, a message is written to the output log, and the exit code is set to 0 (success).

```
final def fileName = props['file']
final def overwrite = props['overwrite']?.toBoolean()
final def contents = props['contents']?:''

try {
    def file = new File(fileName).canonicalFile
    if (file.exists() && !overwrite) {
        println "File $file already exists!"
    }
}
```

```

        System.exit 1
    }
    else {
        file.write(contents)
        println "Successfully ${overwrite?'replaced':'created'} file
            $file with contents:"
        println contents
    }
}
catch (Exception e) {
    println "Error creating file $file: ${e.message}"
    System.exit(1)
}

System.exit(0)

```

## The <post-processing> Element

When a plug-in step's <command> element finishes processing, the step's mandatory <post-processing> element is executed. The <post-processing> element sets the step's output properties (step name/property name, see *IBM uDeploy Properties*) and provides error handling. The <post-processing> element can contain any valid JavaScript script (unlike the <command> element, <post-processing> scripts must be written in JavaScript).

You have access to a `java.util.Properties` variable called `properties`. The `properties` variable has several special properties: `exitCode` contains the process exit code, and `Status` contains the step's status. A `Status` value of `Success` means the step completed successfully. The Create File step's post-processing examines the command's `exitCode` then sets the `Status` property accordingly.

Another available variable—`scanner`—can scan the step's output log and take actions depending on the results. See ??? for an example of scanner usage.

T

You can use a post-processing script to set output properties that can be used in other steps in the same process, which enables complex workflows. Reference prior step output properties this way:

```
${p:stepName/propName}
```

The script defined in the <post-processing> element is the step's default behavior. Users can also provide their own script—overriding the default behavior—when defining the step in the IBM uDeploy editor, see ???.

---

# IBM uDeploy Properties

Properties can be set for the following items:

Also, on any process (component process or application process) you can define properties on the Properties tab to be provided at runtime. - component/application process property

**Table 53. Property Contexts**

Context	Description
environment	<p>Available on the the component's or environment's Properties tab.</p> <p>Referenced: <code>\${p:environment/propertyName}</code>.</p> <p>Both versions use the same syntax. A value set on component environment overrides one with the same name set directly on an environment property. Component environment properties enable you to centralize properties, tracking type and default values, for instance. Environment properties provide ad-hoc lists of property=value pairs.</p>
resource	<p>Selects all the properties with the same value in a given environment.</p>
resource role	<p>Selects all properties with the same value in a given resource.</p>
application	<p>Available on the application's Properties tab (<i>Application</i> &gt; [<i>selected application</i>] &gt; <i>Properties</i>).</p> <p>Referenced: <code>\${p:application/propertyName}</code>.</p>
component	<p>Selects all properties with the same value in a given system.</p>
process	<p>Available on the application's Properties tab (<i>Application</i> &gt; [<i>selected application</i>] &gt; <i>Properties</i>).</p> <p>Referenced: <code>\${p:application/propertyName}</code>.</p> <p>A process step has access to properties set earlier by other steps within the process, to properties set by the application that invoked the component process as well as those on the target environment and resource. Step property values become unavailable once the component process ends.</p>

Context	Description
	Referenced: $\{p:propertyName\}$ .
system	System (global) properties are available on the Settings tab ( <i>Settings &gt; Properties</i> ).  Referenced: $\{p:system/propertyName\}$ .

**Table 54. IBM uDeploy Properties**

Property	Description
version.name	A user defined name to distinguish the version from others. A version name is entered when a new version is imported.
version.id	The number assigned to the version. A version id is created when a new version is imported in CodeStation.
component.name	A user defined name to distinguish it from other components. A component name is entered when creating a new component.
component.id	A unique number IBM uDeploy assigns to distinguish the component from others. The component id is created when a component is created in IBM uDeploy.
resource.name	A user defined name to distinguish it from other resources. The resource name is entered when editing or creating a new resource.
resource.id	A unique number given to a resource. A resource id is assigned when a new resource is created.
application.name	A user defined name to distinguish it from others. An application name is entered when editing or creating a new application.
application.id	A unique number given to an application. An application id is assigned when a new application is created in IBM uDeploy.
environment.name	A user defined name to distinguish the environment from others. An environment name is entered when editing or creating a new environment.
environment.id	A unique number given to an environment. An environment id is assigned when a new environment is created.
agent.id	A unique number IBM uDeploy gives the agent to distinguish it from others with similar names. An agent id is assigned when it is installed on the system.
agent.name	A user defined name to distinguish the agent from others. The agent's name can be

Property	Description
	entered by editing the agent's <code>conf/agent/installed.properties</code> file and restarting the agent.
<code>stepname/propertyname</code>	<p>All steps have the following properties: <i>exitCode</i>, <i>status</i>, <i>lines of interest</i> (LOI—items the post-processing script finds in the step's output log).</p> <p>You can view the properties by using the component's Log pane to examine the step's output log (<i>Components &gt; [selected component] &gt; [View Request action] &gt; [Input/Output Properties action]</i>).</p> <p>Inventory and versions statuses, which are defined with the <i>status</i> property, can be used in application approval gates (see the section called “Application Gates”). The other properties can be used by post-processing scripts, see the section called “The &lt;post-processing&gt; Element”.</p> <p>You can use a post-processing scripts to set output properties that can be used in other steps in the same process, which enables complex workflows. Reference prior step output properties this way:</p> <pre><code>\${p:stepName/propName}</code></pre> <p>To set an environment property from a post-processing script, for example, you set the output property for the step in the post-processing script then use a Set Environment Property step afterwards that consumes the output property.</p>
<code>property_name</code>	Component or application process property; defined on the process's Properties tab. Given value by whoever runs the process.
<code>component/property_name</code>	Component custom property; set on the component's Properties tab.
<code>environment/property_name</code>	Environment property. Defined on the component's or environment's Properties tab. While both use the same syntax, the latter is not associated with any specific component. Values are supplied on the associated environment or component. A value set on component environment overrides one with the same name set directly on an environment property.
<code>resource/property_name</code>	Resource properties. This can include the built-in agent properties as well as any custom properties. Each of these has their own tab on the resource.

Property	Description
resource/role name/property name	Resource role properties. These are defined on resource roles, and the values are set when you add a role to a resource.
application/property name	Application custom properties. These are set on the application's properties tab.
system/property name	Global system properties. These are set on the System Properties tab in the Settings area.

All of the following are comma-separated series of name=value pairs, including each property on the given object. This is useful for token replacement.

**Table 55. Name/Value Pairs**

Property	Description
component/allProperties	Selects all the properties with the same value in a given component.
environment/allProperties	Selects all the properties with the same value in a given environment.
resource/allProperties	Selects all properties with the same value in a given resource.
system/allProperties	Selects all properties with the same value in a given system.

## Using Properties

Properties are referenced with the following format: `${p:property}`.

If, say, you create an environment variable UAT, you would reference it like this:

```
echo ${p:environment/UAT}
```

Output in this case:

UAT

IBM uDeploy escapes the following characters:

\  
=  
,

Replace "\\" with "\"; "\=" with "="; and "\", with ",".

---

# Command Line Client (CLI) Reference

CLI is a command-line interface that provides access to the IBM uDeploy server. It can be used to find or set properties, and perform numerous functions, described below.

To install the tool, download the `udclient.zip` from the IBM uDeploy release page on Supportal (<http://support.UrbanCode, an IBM Company.com>).

## Command Format

To perform a command, open a command window and invoke `udclient` along with the command and parameters. Command's have the following format:

```
udclient [global-args...] [global-flags...] <command> [args...]
```

The global arguments are:

**Table 56.**

Argument	Description
-authtoken, --authtoken	Optional. Can be set via the environment variable DS_AUTH_TOKEN. An authentication token generated by the server. Either an authtoken or a username and password is required.
-password, --password	Optional. Can be set via the environment variable DS_PASSWORD. A password to authenticate with the server. Either an authtoken or a username and password is required.
-username, --username	Optional. Can be set via the environment variable DS_USERNAME. A username to authenticate with the server. Either an authtoken or a username and password is required.
-webserv, --webserv	Required. Can be set via the environment variable DS_WEB_URL. The base URL of the IBM uDeploy server— <code>http://ds.domain.com:8585</code> .

The global flags are:

**Table 57.**

Flag	Description
-t, --getTemplate	Show the JSON template for the command instead of running the command. If a file argument is provided, the template will be output to that file.
-h, --help	Print the full description and help of the given command instead of running the command.
-v, --verbose	Print extra information during execution.

### Note

CLI commands and parameters are case sensitive.



Here is an example using the `getResources` command:

```
udclient -weburl http://localhost:8080 -username admin -password admin  
getResources
```

# Commands

## Note

CLI commands do not support new lines. Entries below are broken for display purposes only.

## addActionToRoleForApplications

Add action to a role for applications.

### Format

```
udclient [global-args...] [global-flags...]  
addActionToRoleForApplications [args...]
```

### Options

```
-role, --role  
    Required. Name of the role  
  
-action, --action  
    Required. Name of the action
```

## addActionToRoleForComponents

Add action to a role for components

### Format

```
udclient [global-args...] [global-flags...]  
addActionToRoleForComponents [args...]
```

### Options

```
-role, --role  
    Required. Name of the role  
  
-action, --action  
    Required. Name of the action
```

## addActionToRoleForEnvironments

Add action to a role for environments

### Format

```
udclient [global-args...] [global-flags...]  
addActionToRoleForEnvironments [args...]
```

### Options

```
-role, --role  
    Required. Name of the role  
  
-action, --action  
    Required. Name of the action
```

## addActionToRoleForResources

Add action to a role for resources

### Format

```
udclient [global-args...] [global-flags...]  
addActionToRoleForResources [args...]
```

### Options

```
-role, --role  
    Required. Name of the role  
  
-action, --action  
    Required. Name of the action
```

## addActionToRoleForUI

Add action to a role for the UI

### Format

```
udclient [global-args...] [global-flags...]  
addActionToRoleForUI [args...]
```

### Options

```
-role, --role  
    Required. Name of the role  
  
-action, --action  
    Required. Name of the action
```

## addAgentToPool

CAdd an agent to an agent pool.

### Format

```
udclient [global-args...] [global-flags...]  
addAgentToPool [args...]
```

### Options

```
-pool, --pool  
    Required. Name or ID of the Agent Pool  
  
-agent, --agent  
    Required. Name or ID of the Agent to add
```

## addComponentToApplication

Add a component to an Application.

### Format

```
udclient [global-args...] [global-flags...]  
addComponentToApplication [args...]
```

### Options

```
-component, --component  
    Required. Name of the component to add  
  
-application, --application
```

Required. Name of the application to add it to.

## addGroupToRoleForApplication

Add a group to a role for an application

### Format

```
udclient [global-args...] [global-flags...]  
addGroupToRoleForApplication [args...]
```

### Options

```
-group, --group  
    Required. Name of the group  
  
-role, --role  
    Required. Name of the role  
  
-application, --application  
    Required. Name of the application
```

## addGroupToRoleForComponent

Add a group to a role for a component

### Format

```
udclient [global-args...] [global-flags...]  
addGroupToRoleForComponent [args...]
```

### Options

```
-group, --group  
    Required. Name of the group  
  
-role, --role  
    Required. Name of the role  
  
-component, --component  
    Required. Name of the component
```

## addGroupToRoleForEnvironment

Add a group to a role for an environment

### Format

```
udclient [global-args...] [global-flags...]  
addGroupToRoleForEnvironment [args...]
```

### Options

```
-group, --group  
    Required. Name of the group  
  
-role, --role  
    Required. Name of the role  
  
-application, --application  
    Required. Name of the application  
  
-environment, --environment  
    Required. Name of the environment
```

## addGroupToRoleForResource

Add a group to a role for a resource

### Format

```
udclient [global-args...] [global-flags...]  
addGroupToRoleForResource [args...]
```

### Options

```
-group, --group  
    Required. Name of the group  
  
-role, --role  
    Required. Name of the role  
  
-resource, --resource  
    Required. Name of the resource
```

## addGroupToRoleForUI

Add a group to a role for the UI

### Format

```
udclient [global-args...] [global-flags...]  
addGroupToRoleForUI [args...]
```

### Options

```
-group, --group  
    Required. Name of the group  
  
-role, --role  
    Required. Name of the role
```

## addLicense

Add a license to the server.

### Format

```
udclient [global-args...] [global-flags...]  
addLicense [args...]
```

### Options

No options for this command.

## addNameConditionToGroup

Add a name condition to a resource group. Only works with dynamic groups.

### Format

```
udclient [global-args...] [global-flags...]  
addNameConditionToGroup [args...]
```

### Options

-comparison, --comparison  
Required. Type of the comparison

-value, --value  
Required. Value of the comparison

-group, --group  
Required. Path of the parent resource group

## addPropertyConditionToGroup

Add a property condition to a resource group. Only works with dynamic groups.

### Format

```
udclient [global-args...] [global-flags...]
addPropertyConditionToGroup [args...]
```

### Options

-property, --property  
Required. Name of the property

-comparison, --comparison  
Required. Type of the comparison

-value, --value  
Required. Value of the comparison

-group, --group  
Required. Path of the parent resource group

## addResourceToGroup

Add a resource to a resource group. Only works with static groups.

### Format

```
udclient [global-args...] [global-flags...]
addResourceToGroup [args...]
```

### Options

```
-resource, --resource  
    Required. Name of the resource to add  
  
-group, --group  
    Required. Path of the resource group to add to
```

## addRoleToResource

Add a role to a resource.

### Format

```
udclient [global-args...] [global-flags...]  
addRoleToResource [args...]
```

### Options

```
-resource, --resource  
    Required. Name of the parent resource.  
  
-role, --role  
    Required. Name of the new resource.
```

## addRoleToResourceWithProperties

Add a role to a resource. This command takes a JSON request body. Use the -t flag to view the template for the data required for this command.

### Format

```
udclient [global-args...] [global-flags...]  
addRoleToResourceWithProperties [args...] [-] [filename]
```

```
-  
    Read JSON input from the stdin. See command for requirements.
```

```
filename  
    Read JSON input from a file with the given filename. See command for requirements.
```

### Options



No options for this command.

## addUserToGroup

Add a user to a group

### Format

```
udclient [global-args...] [global-flags...]  
addUserToGroup [args...]
```

### Options

```
-user, --user  
    Required. Name of the user  
  
-group, --group  
    Required. Name of the group
```

## addUserToRoleForApplication

Add a user to a role for an application

### Format

```
udclient [global-args...] [global-flags...]  
addUserToRoleForApplication [args...]
```

### Options

```
-user, --user  
    Required. Name of the user  
  
-role, --role  
    Required. Name of the role  
  
-application, --application  
    Required. Name of the application
```

## addUserToRoleForComponent

Add a user to a role for a component

### Format

```
udclient [global-args...] [global-flags...]  
addUserToRoleForComponent [args...]
```

### Options

```
-user, --user  
    Required. Name of the user  
  
-role, --role  
    Required. Name of the role  
  
-component, --component  
    Required. Name of the component
```

## addUserToRoleForEnvironment

Add a user to a role for an environment

### Format

```
udclient [global-args...] [global-flags...]  
addUserToRoleForEnvironment [args...]
```

### Options

```
-user, --user  
    Required. Name of the user  
  
-role, --role  
    Required. Name of the role  
  
-application, --application  
    Required. Name of the application  
  
-environment, --environment  
    Required. Name of the environment
```

## addUserToRoleForResource

Add a user to a role for a resource

### Format

```
udclient [global-args...] [global-flags...]  
addUserToRoleForResource [args...]
```

### Options

```
-user, --user  
    Required. Name of the user  
  
-role, --role  
    Required. Name of the role  
  
-resource, --resource  
    Required. Name of the resource
```

## addUserToRoleForUI

Add a user to a role for the UI

### Format

```
udclient [global-args...] [global-flags...]  
addUserToRoleForUI [args...]
```

### Options

```
-user, --user  
    Required. Name of the user  
  
-role, --role  
    Required. Name of the role
```

## addVersionFiles

Upload files to a version

### Format

```
udclient [global-args...] [global-flags...]  
addVersionFiles [args...]
```

### Options

`-component, --component`  
Optional. Name/ID of the component (Only required if not using version ID)

`-version, --version`  
Required. Name/ID of the version

`-base, --base`  
Required. Local base directory for upload. All files inside this will be sent.

`-offset, --offset`  
Optional. Target path offset (the directory in the version files to which these files should be added)

## addVersionStatus

Add a status to a version

### Format

```
udclient [global-args...] [global-flags...]
addVersionStatus [args...]
```

### Options

`-component, --component`  
Optional. Name/ID of the component (Only required if not using version ID)

`-version, --version`  
Required. Name/ID of the version

`-status, --status`  
Required. Name of the status to apply

## createAgentPool

Create an agent pool. This command takes a JSON request body. Use the `-t` flag to view the template for the data required for this command.

### Format

```
udclient [global-args...] [global-flags...]  
createAgentPool [args...]
```

### Options

No options for this command.

## createApplication

Create a new application. This command takes a JSON request body. Use the -t flag to view the template for the data required for this command.

### Format

```
udclient [global-args...] [global-flags...]  
createApplication [args...] [-] [filename]
```

-

Read JSON input from the stdin. See command for requirements.

filename

Read JSON input from a file with the given filename. See command for requirements.

### Options

No options for this command.

## createApplicationProcess

Create a new application process. This command takes a JSON request body. Use the -t flag to view the template for the data required for this command.

### Format

```
udclient [global-args...] [global-flags...]  
createApplicationProcess [args...] [-] [filename]
```

-

Read JSON input from the stdin. See command for requirements.

filename

Read JSON input from a file with the given filename. See command for requirements.

### Options

No options for this command.

## createComponent

Create a new component. This command takes a JSON request body. Use the -t flag to view the template for the data required for this command.

### Format

```
udclient [global-args...] [global-flags...]
createComponent [args...] [-] [filename]
```

-

Read JSON input from the stdin. See command for requirements.

filename

Read JSON input from a file with the given filename. See command for requirements.

### Options

No options for this command.

## createComponentProcess

Create a new component process. This command takes a JSON request body. Use the -t flag to view the template for the data required for this command.

### Format

```
udclient [global-args...] [global-flags...]
createComponentProcess [args...] [-] [filename]
```

-

Read JSON input from the stdin. See command for requirements.

filename

Read JSON input from a file with the given filename. See command for requirements.

### Options

No options for this command.

## createDynamicResourceGroup

Create a new static resource group.

### Format

```
udclient [global-args...] [global-flags...]
createDynamicResourceGroup [args...]
```

### Options

```
-path, --path
    Required. Path to add the resource group to (parent resource group
    path).

-name, --name
    Required. Name of the new resource group.
```

## createEnvironment

Create a new environment.

### Format

```
udclient [global-args...] [global-flags...]
createEnvironment [args...]
```

### Options

```
-application, --application
    Required. Application to add the environment to.

-name, --name
    Required. Name of the new environment.

-description, --description
    Optional. Description of the new environment.

-color, --color
    Optional. Color of the new environment.

-requireApprovals, --requireApprovals
    Optional. Does the environment require approvals?
```

## createGroup

Add a new group

### Format

```
udclient [global-args...] [global-flags...] createGroup [args...]
```

### Options

```
-group, --group  
    Required. Name of the group
```

## createMapping

Create a new mapping.

### Format

```
udclient [global-args...] [global-flags...]  
createMapping [args...]
```

### Options

```
-environment, --environment  
    Required. The environment for the mapping.  
  
-component, --component  
    Required. The component for the mapping.  
  
-resourceGroupPath, --resourceGroupPath  
    Required. The resource group for the mapping.  
  
-application, --application  
    Optional. The application for the mapping. Only necessary if  
    specifying env name instead of id.
```

## createResource

Create a resource.



### Format

```
udclient [global-args...] [global-flags...]  
createResource [args...]
```

### Options

```
-parentAgent, --parentAgent  
    Optional. Name or ID of the parent agent.(One of parentAgent,  
    parentResource, parentAgentPool or source must be specified)  
  
-parentResource, --parentResource  
    Optional. Name or ID of the parent resource or agent.  
  
-parentAgentPool, --parentAgentPool  
    Optional. Name or ID of the parent agent pool.  
  
-name, --name  
    Required. Name of the new resource.  
  
-description, --description  
    Optional. Description of the resource.  
  
-source, --source  
    Optional. Name of a subresource to copy.
```

## createResourceGroup

Create a new static resource group.

### Format

```
udclient [global-args...] [global-flags...]  
createResourceGroup [args...]
```

### Options

```
-path, --path  
    Required. Path to add the resource group to (parent resource group  
    path).  
  
-name, --name  
    Required. Name of the new resource group.
```

## createRoleForApplications

Create a role for applications

### Format

```
udclient [global-args...] [global-flags...]  
createRoleForApplications [args...]
```

### Options

```
-role, --role  
    Required. Name of the role
```

## createRoleForComponents

Create a role for components

### Format

```
udclient [global-args...] [global-flags...]  
createRoleForComponents [args...]
```

### Options

```
-role, --role  
    Required. Name of the role
```

## createRoleForEnvironments

Create a role for environments

### Format

```
udclient [global-args...] [global-flags...]  
createRoleForEnvironments [args...]
```

### Options

```
-role, --role  
    Required. Name of the role
```

## createRoleForResources

Create a role for resources

### Format

```
udclient [global-args...] [global-flags...]  
createRoleForResources [args...]
```

### Options

```
-role, --role  
    Required. Name of the role
```

## createRoleForUI

Create a role for the UI

### Format

```
udclient [global-args...] [global-flags...]  
createRoleForUI [args...]
```

### Options

```
-role, --role  
    Required. Name of the role
```

## createSubresource

Create a new subresource.

### Format

```
udclient [global-args...] [global-flags...]  
createSubresource [args...]
```

### Options

-parent, --parent  
Required. Name of the parent resource.

-name, --name  
Required. Name of the new resource.

-description, --description  
Optional. Description of the resource.

## createUser

Add a new user This command takes a JSON request body. Use the -t flag to view the template for the data required for this command.

### Format

```
udclient [global-args...] [global-flags...]
createUser [args...] [-] [filename]
```

-

Read JSON input from the stdin. See command for requirements.

filename

Read JSON input from a file with the given filename. See command for requirements.

### Options

No options for this command.

## createVersion

Create a new version for a component

### Format

```
udclient [global-args...] [global-flags...]
createVersion [args...]
```

### Options

-component, --component  
Required. Name/ID of the component

-name, --name  
Required. Name of the new version

## deleteAgent

Remove an agent.

### Format

```
udclient [global-args...] [global-flags...]  
deleteAgent [args...]
```

### Options

-agent, --agent  
Required. Name or ID of the agent

## deleteAgentPool

Remove an agent pool.

### Format

```
udclient [global-args...] [global-flags...]  
deleteAgentPool [args...]
```

### Options

-pool, --pool  
Required. Name or ID of the agent pool

## deleteGroup

Delete a group

### Format

```
udclient [global-args...] [global-flags...]  
deleteGroup [args...]
```

### Options

-group, --group  
Required. Name of the group

## deleteResource

Remove a resource.

### Format

```
udclient [global-args...] [global-flags...]  
deleteResource [args...]
```

### Options

```
-resource, --resource  
    Required. Name of the resource to configure
```

## deleteResourceGroup

null

### Format

```
udclient [global-args...] [global-flags...]  
deleteResourceGroup [args...]
```

### Options

```
-group, --group  
    Required. Path of the resource group to delete
```

## deleteResourceProperty

Remove a custom property from a resource

### Format

```
udclient [global-args...] [global-flags...]  
deleteResourceProperty [args...]
```

### Options

```
-resource, --resource  
    Required. Name of the resource to configure
```

`-name, --name`  
Required. Name of the property

## deleteUser

Delete a user

### Format

```
udclient [global-args...] [global-flags...]
deleteUser [args...]
```

### Options

`-user, --user`  
Required. Name of the user

## exportGroup

Add a new group

### Format

```
udclient [global-args...] [global-flags...]
exportGroup [args...]
```

### Options

`-group, --group`  
Required. Name of the group

## getAgent

Get a JSON representation of an agent.

### Format

```
udclient [global-args...] [global-flags...]
getAgent [args...]
```

### Options

`-agent, --agent`  
Required. Name of the Agent Pool to look up

## getAgentPool

Get a JSON representation of an agent pool.

### Format

```
udclient [global-args...] [global-flags...]
getAgentPool [args...]
```

### Options

`-pool, --pool`  
Required. Name of the Agent Pool to look up

## getAgentPools

Get a JSON array of all agent pools.

### Format

```
udclient [global-args...] [global-flags...]
getAgentPools [args...]
```

### Options

`-active, --active`  
Optional. List active agent pools - Default true

`-inactive, --inactive`  
Optional. List inactive agent pools - Default true

## getAgents

Get a JSON array of all agents.

### Format

```
udclient [global-args...] [global-flags...]
getAgents [args...]
```

### Options

`-active, --active`



Optional. List active agents - Default true

-inactive, --inactive

Optional. List inactive agents - Default false

## getApplication

Get a JSON representation of an application

### Format

```
udclient [global-args...] [global-flags...]
getApplication [args...]
```

### Options

-application, --application

Required. Name of the application to look up

## getApplicationProcess

Get a JSON representation of an Application Process

### Format

```
udclient [global-args...] [global-flags...]
getApplicationProcess [args...]
```

### Options

-application, --application

Required. Name of the application

-applicationProcess, --applicationProcess

Required. Name of the process

## getApplicationProcessRequestStatus

Get the status of an application process.

### Format

```
udclient [global-args...] [global-flags...]  
getApplicationProcessRequestStatus [args...]
```

### Options

```
-request, --request  
    Required. ID of the application process request to view
```

## getApplicationProperties

Get the values of custom properties for an application.

### Format

```
udclient [global-args...] [global-flags...]  
getApplicationProperties [args...]
```

### Options

```
-application, --application  
    Required. Name or id of the application
```

## getApplicationProperty

Get the value of custom property for an application.

### Format

```
udclient [global-args...] [global-flags...]  
getApplicationProperty [args...]
```

### Options

```
-application, --application  
    Required. Name or id of the application  
  
-name, --name  
    Required. Name of the property
```

## getApplications

Get a JSONArray representation of all applications

### Format

```
udclient [global-args...] [global-flags...]  
getApplications [args...]
```

### Options

No options for this command.

## getComponent

Get a JSON representation of a component

### Format

```
udclient [global-args...] [global-flags...]  
getComponent [args...]
```

### Options

```
-component, --component  
    Required. Name of the component to look up
```

## getComponentEnvironmentProperties

Get all values of custom properties for a component.

### Format

```
udclient [global-args...] [global-flags...]  
getComponentEnvironmentProperties [args...]
```

### Options

```
-component, --component  
    Required. Name or id of the component  
  
-environment, --environment  
    Required. Name or id of the environment  
  
-application, --application  
    Optional. Name or id of the application
```

## getComponentEnvironmentProperty

Get the value of a custom property on a component.

#### Format

```
udclient [global-args...] [global-flags...]  
getComponentEnvironmentProperty [args...]
```

#### Options

-name, --name  
Required. Name of the property to look up

-component, --component  
Required. Name or id of the component

-environment, --environment  
Required. Name or id of the environment

-application, --application  
Optional. Name or id of the application

## getComponentProcess

Get a JSON representation of a componentProcess

#### Format

```
udclient [global-args...] [global-flags...]  
getComponentProcess [args...]
```

#### Options

-component, --component  
Required. Name of the component

-componentProcess, --componentProcess  
Required. Name of the component

## getComponents

Get a JSONArray representation of all components

#### Format

```
udclient [global-args...] [global-flags...]  
getComponents [args...]
```

### Options

No options for this command.

## getComponentsInApplication

Get all components in an application

### Format

```
udclient [global-args...] [global-flags...]  
getComponentsInApplication [args...]
```

### Options

```
-application, --application  
    Required. Name of the application to get components for
```

## GetComponentProperties

Get the values of all of a component's custom properties.

### Format

```
udclient [global-args...] [global-flags...]  
GetComponentProperties [args...]
```

### Options

```
-component, --component  
    Required. Name of the component
```

## GetComponentProperty

Get the value of a component's custom property.

### Format

```
udclient [global-args...] [global-flags...]  
GetComponentProperty [args...]
```

### Options

`-component, --component`  
Required. Name of the component

`-name, --name`  
Required. Name of the property

## getEnvironment

Get a JSON representation of an environment

### Format

```
udclient [global-args...] [global-flags...]
getEnvironment [args...]
```

### Options

`-environment, --environment`  
Required. Name of the environment to look up

## getEnvironmentProperties

Get the value of all custom properties for an environment.

### Format

```
udclient [global-args...] [global-flags...]
getEnvironmentProperties [args...]
```

### Options

`-environment, --environment`  
Required. Name or id of the environment

`-application, --application`  
Optional. Name or id of the application (required unless environment id is specified)

## getEnvironmentProperty

Get the value of a component's custom propert.

### Format

```
udclient [global-args...] [global-flags...]  
getEnvironmentProperty [args...]
```

### Options

```
-name, --name  
    Required. Name of the property to look up  
  
-environment, --environment  
    Required. Name or id of the environment  
  
-application, --application  
    Optional. Name or id of the application (required unless environment  
    id is specified)
```

## getEnvironmentsInApplication

Get all environments for an application.

### Format

```
udclient [global-args...] [global-flags...]  
getEnvironmentsInApplication [args...]
```

### Options

```
-application, --application  
    Required. Name of the application to get environments for  
  
-active, --active  
    Optional. List active environments - Default true  
  
-inactive, --inactive  
    Optional. List inactive environments - Default false
```

## getGroupsForResource

Get a JSON array representation of all the groups to which a resource belongs.

### Format

```
udclient [global-args...] [global-flags...]  
getGroupsForResource [args...]
```

### Options

```
-resource, --resource
```

Required. Name/id of the resource.

## getMapping

Get a JSON representation of a mapping

### Format

```
udclient [global-args...] [global-flags...]
getMapping [args...]
```

### Options

```
-mapping, --mapping
    Required. ID of the mapping to look up
```

## getMappingsForApplicationEnvironment

Get the component mappings for an application environment.

### Format

```
udclient [global-args...] [global-flags...]
getMappingsForApplicationEnvironment [args...]
```

### Options

```
-environment, --environment
    Required. Name or ID of the environment to look up

-application, --application
    Optional. Name of the application - required if using environment
    name instead of ID.
```

## getMappingsForGroup

Get the component environment mappings for a resource group.

### Format

```
udclient [global-args...] [global-flags...]
getMappingsForGroup [args...]
```

### Options



`-group, --group`

Required. Path of the resource group to get mappings for

## getResource

Get a JSON representation of a resource

### Format

```
udclient [global-args...] [global-flags...]
getResource [args...]
```

### Options

`-resource, --resource`

Required. Name of the resource to look up

## getResourceGroup

Get a JSON representation of a resource group

### Format

```
udclient [global-args...] [global-flags...]
getResourceGroup [args...]
```

### Options

`-group, --group`

Required. Path of the resource group to show

## getResourceGroups

Get a JSONArray representation of all resource groups

### Format

```
udclient [global-args...] [global-flags...]
getResourceGroups [args...]
```

### Options

No options for this command.

## getResourceProperties

Get all property values for a resource.

### Format

```
udclient [global-args...] [global-flags...]  
getResourceProperties [args...]
```

### Options

```
-resource, --resource  
    Required. Name/id of the resource
```

## getResourceProperty

Get the value of a custom property on a resource

### Format

```
udclient [global-args...] [global-flags...]  
getResourceProperty [args...]
```

### Options

```
-resource, --resource  
    Required. Name of the resource  
  
-name, --name  
    Required. Name of the property
```

## getResources

Get a JSONArray representation of all resources

### Format

```
udclient [global-args...] [global-flags...]  
getResources [args...]
```

### Options

No options for this command.

## getResourcesInGroup

Get a JSONArray representation of all resources in a group

### Format

```
udclient [global-args...] [global-flags...]  
getResourcesInGroup [args...]
```

### Options

```
-group, --group  
    Required. Path of the resource group
```

## getResourceSecurity

Get a list of security roles and members for a resource.

### Format

```
udclient [global-args...] [global-flags...]  
getResourceSecurity [args...]
```

### Options

```
-resource, --resource  
    Required. Name/id of the resource
```

## getRoleForApplications

Get a JSON representation of a role

### Format

```
udclient [global-args...] [global-flags...]  
getRoleForApplications [args...]
```

### Options

`-role, --role`  
Required. Name of the role

## getRoleForComponents

Get a JSON representation of a role

### Format

```
udclient [global-args...] [global-flags...]  
getRoleForComponents [args...]
```

### Options

`-role, --role`  
Required. Name of the role

## getRoleForEnvironments

Get a JSON representation of a role

### Format

```
udclient [global-args...] [global-flags...]  
getRoleForEnvironments [args...]
```

### Options

`-role, --role`  
Required. Name of the role

## getRoleForResources

Get a JSON representation of a role

### Format

```
udclient [global-args...] [global-flags...]
```

```
getRoleForResources [args...]
```

#### Options

```
-role, --role  
    Required. Name of the role
```

## getRoleForUI

Get a JSON representation of a role

#### Format

```
udclient [global-args...] [global-flags...]  
getRoleForUI [args...]
```

#### Options

```
-role, --role  
    Required. Name of the role
```

## getRolesForResource

Get a list of roles for a resource.

#### Format

```
udclient [global-args...] [global-flags...]  
getRolesForResource [args...]
```

#### Options

```
-resource, --resource  
    Required. Name/id of the resource
```

## getSystemProperties

Get all system property values.

#### Format

```
udclient [global-args...] [global-flags...]  
getSystemProperties [args...]
```

#### Options

No options.

## getSystemProperty

Get a system property value.

#### Format

```
udclient [global-args...] [global-flags...]  
getSystemProperty [args...]
```

#### Options

`-name, --name`  
Required. Name of the property

## getUser

Get a JSON representation of a user

#### Format

```
udclient [global-args...] [global-flags...]  
getUser [args...]
```

#### Options

`-user, --user`  
Required. Name of the user

## importGroup

Add a new group This command takes a JSON request body. Use the `-t` flag to view the template for the data required for this command.

#### Format

```
udclient [global-args...] [global-flags...]  
importGroup [args...] [-] [filename]
```

-  
Read JSON input from the stdin. See command for requirements.

filename  
Read JSON input from a file with the given filename. See command for requirements.

### Options

No options for this command.

## importVersions

Run the source config integration for a component This command takes a JSON request body. Use the -t flag to view the template for the data required for this command.

### Format

```
udclient [global-args...] [global-flags...]
importVersions [args...] [-] [filename]
```

-  
Read JSON input from the stdin. See command for requirements.

filename  
Read JSON input from a file with the given filename. See command for requirements.

### Options

No options for this command.

## inactivateEnvironment

Set an environment to inactive.

### Format

```
udclient [global-args...] [global-flags...]
inactivateEnvironment [args...]
```

### Options

-environment, --environment

Required. Name or ID of the environment to look up

-application, --application

Optional. Name of the application - required if using environment name instead of ID.

## installAgent

Install an agent.

### Format

```
udclient [global-args...] [global-flags...]
installAgent [args...]
```

### Options

-name, --name

Required. Name of the agent resource

-host, --host

Required. Host name or IP of the agent machine

-port, --port

Required. SSH port (22) of the agent machine

-sshUsername, --sshUsername

Required. Username to use to ssh to the agent machine

-sshPassword, --sshPassword

Optional. Password to use to ssh to the agent machine (exclude this flag to use Public Key Authentication instead)

-installDir, --installDir

Required. Installation directory of the agent

-javaHomePath, --javaHomePath

Required. Path to Java on the agent machine

-tempDirPath, --tempDirPath

Required. Path to directory to install from on the agent machine

-serverHost, --serverHost

Required. Host name or IP of the uDeploy server or Agent Relay the agent should connect to

-serverPort, --serverPort

Required. Agent communication port of the uDeploy server (7918) or Agent Relay (7916) the agent should connect to

-proxyHost, --proxyHost

Optional. Host name or IP of the Agent Relay if used



`-proxyPort, --proxyPort`  
Optional. HTTP proxy port of the Agent Relay if used (20080)

`-mutualAuth, --mutualAuth`  
Optional. True if the agent should enforce certificate validation for mutual authentication

## login

Login for further requests

### Format

```
udclient [global-args...] [global-flags...] login [args...]
```

### Options

No options for this command.

## logout

Logout

### Format

```
udclient [global-args...] [global-flags...]
logout [args...]
```

### Options

No options for this command.

## removeActionFromRoleForApplications

Add action to a role for applications

### Format

```
udclient [global-args...] [global-flags...]
removeActionFromRoleForApplications [args...]
```

### Options

```
-role, --role  
    Required. Name of the role  
  
-action, --action  
    Required. Name of the action
```

## removeActionFromRoleForComponents

Add action to a role for components

### Format

```
udclient [global-args...] [global-flags...]  
removeActionFromRoleForComponents [args...]
```

### Options

```
-role, --role  
    Required. Name of the role  
  
-action, --action  
    Required. Name of the action
```

## removeActionFromRoleForEnvironments

Add action to a role for environments

### Format

```
udclient [global-args...] [global-flags...]  
removeActionFromRoleForEnvironments [args...]
```

### Options

```
-role, --role  
    Required. Name of the role  
  
-action, --action  
    Required. Name of the action
```

## removeActionFromRoleForResources

Add action to a role for resources

### Format

```
udclient [global-args...] [global-flags...]  
removeActionFromRoleForResources [args...]
```

### Options

```
-role, --role  
    Required. Name of the role  
  
-action, --action  
    Required. Name of the action
```

## removeActionFromRoleForUI

Add action to a role for the UI

### Format

```
udclient [global-args...] [global-flags...]  
removeActionFromRoleForUI [args...]
```

### Options

```
-role, --role  
    Required. Name of the role  
  
-action, --action  
    Required. Name of the action
```

## removeAgentFromPool

Remove an agent from an agent pool.

### Format

```
udclient [global-args...] [global-flags...]
```

```
removeAgentFromPool [args...]
```

#### Options

```
-pool, --pool
```

Required. Name or ID of the Agent Pool

```
-agent, --agent
```

Required. Name or ID of the Agent to remove

## removeGroupFromRoleForApplication

Remove a group to a role for an application

#### Format

```
udclient [global-args...] [global-flags...]  
removeGroupFromRoleForApplication [args...]
```

#### Options

```
-group, --group
```

Required. Name of the group

```
-role, --role
```

Required. Name of the role

```
-application, --application
```

Required. Name of the application

## removeGroupFromRoleForComponent

Remove a group to a role for a component

#### Format

```
udclient [global-args...] [global-flags...]  
removeGroupFromRoleForComponent [args...]
```

#### Options

```
-group, --group
```

Required. Name of the group

-role, --role  
Required. Name of the role

-component, --component  
Required. Name of the component

## removeGroupFromRoleForEnvironment

Remove a group to a role for an environment

### Format

```
udclient [global-args...] [global-flags...]
removeGroupFromRoleForEnvironment [args...]
```

### Options

-group, --group  
Required. Name of the group

-role, --role  
Required. Name of the role

-application, --application  
Required. Name of the application

-environment, --environment  
Required. Name of the environment

## removeGroupFromRoleForResource

Remove a group to a role for a resource

### Format

```
udclient [global-args...] [global-flags...]
removeGroupFromRoleForResource [args...]
```

### Options

-group, --group  
Required. Name of the group

`-role, --role`  
Required. Name of the role

`-resource, --resource`  
Required. Name of the resource

## removeGroupFromRoleForUI

Remove a group to a role for the UI

### Format

```
udclient [global-args...] [global-flags...]
removeGroupFromRoleForUI [args...]
```

### Options

`-group, --group`  
Required. Name of the group

`-role, --role`  
Required. Name of the role

## removeMapping

Remove a mapping.

### Format

```
udclient [global-args...] [global-flags...]
removeMapping [args...]
```

### Options

`-environment, --environment`  
Required. The environment for the mapping.

`-component, --component`  
Required. The component for the mapping.

`-resourceGroupPath, --resourceGroupPath`  
Optional. The resource group path for the mapping, if not using a resource.

`-resource, --resource`  
Optional. The resource for the mapping, if not using a group.

`-application, --application`  
Optional. The application for the mapping. Only necessary if specifying env name instead of id.

## removeResourceFromGroup

Remove a resource from a resource group. Only works with static groups.

### Format

```
udclient [global-args...] [global-flags...]
removeResourceFromGroup [args...]
```

### Options

`-resource, --resource`  
Required. Name of the resource to remove

`-group, --group`  
Required. Path of the resource group to remove from

## removeRoleForApplications

Create a role for applications

### Format

```
udclient [global-args...] [global-flags...]
removeRoleForApplications [args...]
```

### Options

`-role, --role`  
Required. Name of the role

## removeRoleForComponents

Create a role for components

### Format

```
udclient [global-args...] [global-flags...]  
removeRoleForComponents [args...]
```

### Options

```
-role, --role  
    Required. Name of the role
```

## removeRoleForEnvironments

Create a role for environments

### Format

```
udclient [global-args...] [global-flags...]  
removeRoleForEnvironments [args...]
```

### Options

```
-role, --role  
    Required. Name of the role
```

## removeRoleForResources

Create a role for resources

### Format

```
udclient [global-args...] [global-flags...]  
removeRoleForResources [args...]
```

### Options

```
-role, --role  
    Required. Name of the role
```



## removeRoleForUI

Create a role for the UI

### Format

```
udclient [global-args...] [global-flags...]
removeRoleForUI [args...]
```

### Options

```
-role, --role
    Required. Name of the role
```

## removeRoleFromResource

Remove a role from a resource.

### Format

```
udclient [global-args...] [global-flags...]
removeRoleFromResource [args...]
```

### Options

```
-resource, --resource
    Required. Name of the parent resource.

-role, --role
    Required. Name of the new resource.
```

## removeUserFromGroup

Remove a user from a group

### Format

```
udclient [global-args...] [global-flags...]
removeUserFromGroup [args...]
```

### Options

```
-user, --user  
    Required. Name of the user  
  
-group, --group  
    Required. Name of the group
```

## removeUserFromRoleForApplication

Remove a user to a role for an application

### Format

```
udclient [global-args...] [global-flags...]  
removeUserFromRoleForApplication [args...]
```

### Options

```
-user, --user  
    Required. Name of the user  
  
-role, --role  
    Required. Name of the role  
  
-application, --application  
    Required. Name of the application
```

## removeUserFromRoleForComponent

Remove a user to a role for a component

### Format

```
udclient [global-args...] [global-flags...]  
removeUserFromRoleForComponent [args...]
```

### Options

```
-user, --user  
    Required. Name of the user
```

-role, --role  
Required. Name of the role

-component, --component  
Required. Name of the component

## removeUserFromRoleForEnvironment

Remove a user to a role for an environment

### Format

```
udclient [global-args...] [global-flags...] removeUserFromRoleForEnvironment
```

### Options

-user, --user  
Required. Name of the user

-role, --role  
Required. Name of the role

-application, --application  
Required. Name of the application

-environment, --environment  
Required. Name of the environment

## removeUserFromRoleForResource

Remove a user to a role for a resource

### Format

```
udclient [global-args...] [global-flags...]  
removeUserFromRoleForResource [args...]
```

### Options

-user, --user  
Required. Name of the user

-role, --role  
Required. Name of the role

-resource, --resource  
Required. Name of the resource

## removeUserFromRoleForUI

Remove a user to a role for the UI

### Format

```
udclient [global-args...] [global-flags...]
removeUserFromRoleForUI [args...]
```

### Options

-user, --user  
Required. Name of the user

-role, --role  
Required. Name of the role

## removeVersionStatus

Remove a status from a version.

### Format

```
udclient [global-args...] [global-flags...]
removeVersionStatus [args...]
```

### Options

-component, --component  
Optional. Name/ID of the component (Only required if not using version ID)

-version, --version  
Required. Name/ID of the version

-status, --status  
Required. Name of the status to apply

## repeatApplicationProcessRequest

Repeat an application process request.

### Format

```
udclient [global-args...] [global-flags...]
repeatApplicationProcessRequest [args...]
```

### Options

```
-request, --request
    Required. ID of the application process request to repeat
```

## requestApplicationProcess

Submit an application process request to run immediately. This command takes a JSON request body. Use the -t flag to view the template for the data required for this command.

### Format

```
udclient [global-args...] [global-flags...]
requestApplicationProcess [args...] [-] [filename]
```

```
-
    Read JSON input from the stdin. See command for requirements.
```

```
filename
    Read JSON input from a file with the given filename. See command for requirements.
```

### Options

```
No options for this command.
```

## restartAgent

Restart an agent

### Format

```
udclient [global-args...] [global-flags...] restartAgent [args...]
```

### Options

-agent, --agent  
Required. Name / ID of the Agent to restart

## setApplicationProperty

Set property on an application.

### Format

```
udclient [global-args...] [global-flags...]
setApplicationProperty [args...]
```

### Options

-name, --name  
Required. Name of the property to set

-value, --value  
Optional. Value of the property to set

-isSecure, --isSecure  
Optional. Defaults to current state of property, or false if property is not yet defined

-application, --application  
Required. Name of the application to look up

## setComponentProperty

Set property on component

### Format

```
udclient [global-args...] [global-flags...]
setComponentProperty [args...]
```

### Options

-propName, --propName

Required. Name of the property to set

-propValue, --propValue

Required. Value of the property to set

-component, --component

Required. Name of the component to look up

## setComponentEnvironmentProperty

Set property on component/environment mapping

### Format

```
udclient [global-args...] [global-flags...]
setComponentEnvironmentProperty [args...]
```

### Options

-name, --name

Required. Name of the property to set

-value, --value

Required. Value of the property to set

-component, --component

Required. Name of the component to look up

-environment, --environment

Required. Name or id of the environment to look up

-isSecure, --isSecure

Optional. Defaults to property state, or false if not defined

-application, --application

Optional. Name/ID of the application (required if referencing environment by name)

## setEnvironmentProperty

Set property on an environment

### Format

```
udclient [global-args...] [global-flags...] setEnvironmentProperty [args...]
```

### Options

`-name, --name`  
Required. Name of the property to set

`-value, --value`  
Optional. Value of the property to set

`-environment, --environment`  
Required. Name/ID of the environment to look up

`-isSecure, --isSecure`  
Optional. Defaults to current state of property, or false if property is not yet defined.

`-application, --application`  
Optional. Name/ID of the application (required if referencing environment by name)

## setResourceProperty

Set a custom property on a resource

### Format

```
udclient [global-args...] [global-flags...]
setResourceProperty [args...]
```

### Options

`-resource, --resource`  
Required. Name of the resource to configure

`-name, --name`  
Required. Name of the property

`-value, --value`  
Optional. New value for the property

## setSystemProperty

Set a property on the system.



### Format

```
udclient [global-args...] [global-flags...]  
setSystemProperty [args...]
```

### Options

```
-name, --name  
    Required. Name of the property to set  
  
-isSecure, --isSecure  
    Optional. Defaults to current state of property, or false if  
    property is not yet defined.  
  
-value, --value  
    Optional. Value of the property to set
```

## shutdownAgent

Shut down an agent.

### Format

```
udclient [global-args...] [global-flags...]  
shutdownAgent [args...]
```

### Options

```
-agent, --agent  
    Required. Name / ID of the Agent to shut down
```

## testAgent

Test an agent.

### Format

```
udclient [global-args...] [global-flags...]  
testAgent [args...]
```

### Options

```
-agent, --agent  
    Required. Name / ID of the Agent to shut down
```

## updateUser

Add a new user This command takes a JSON request body. Use the -t flag to view the template for the data required for this command.

### Format

```
udclient [global-args...] [global-flags...]  
updateUser [args...] [-] [filename]
```

-

Read JSON input from the stdin. See command for requirements.

filename

Read JSON input from a file with the given filename. See command for requirements.

### Options

-user, --user

Required. Name of the user

---

# Best Practices

## General Practices

### Tip

Rule #1: Keep things as simple as possible. Complicated designs make the system more fragile and problems harder to troubleshoot.

#### **Incorporate error handling in process designs.**

Make error handling a routine part of your process designs. For example, check to ensure a required link exists, and if it doesn't, create it. Normally you might never create the link and so have no interest in testing that logic, but if the link fails on a production machine, the entire deployment might fail. Instead of risking that, remove the link and re-create it every time.

#### **Avoid incremental deployments.**

While incremental deployments can make sense for manual deployments, tracking incremental versions can quickly become a nightmare with automated deployments. If you are deploying 100 files, for example, deploy all of them every time and let IBM uDeploy figure out which ones have actually changed. This way, if a file was unexpectedly changed, the deployment process itself will fix it.

#### **Consider how your processes would run on a new machine when designing them.**

The goal of this exercise is to guarantee the state of your environments by being able to destroy a VM/image and rebuild it as part of a routine deployment.

#### **Replace custom scripts with IBM uDeploy plug-ins wherever possible.**

By replacing scripts that are scattered over a large number of machines with plug-ins, you reduce maintenance and gain instant functionality whenever a plug-in is upgraded. Most of IBM uDeploy's plug-ins are OS independent which provides better re-usability. Plug-ins make updating much easier because all steps using a plug-in are updated at once.

### Tip

Improve current processes when preparing to migrate to IBM uDeploy. Frequently, pre-IBM uDeploy processes were designed around limitations that IBM uDeploy is designed to eliminate. Think about a perfect-world process and then attempt to implement that in IBM uDeploy.

#### **Establish naming conventions that promote reuse.**

If teams know they can always access a property called, say, *db.url* to get a database connection string, there is no need to define a new property every time. And if there is a common component that is reusable, name it in a way other teams can find it rather than creating a copy.

#### **Avoid process steps that do nothing.**

Steps intended to sleep or synchronize with other systems are difficult to abort and can become stuck. Use the locking steps instead, and split large processes into smaller ones and orchestrate those.

**Use the same processes across all environments.**

This ensures that preproduction deployments provide reliable results. Get around environment differences by using environment properties and components mapped to specific environments. A production environment might have a monitoring system, for example, and a component that disables/enables monitoring can be mapped specifically to the production environment.

**When adding component versions, use the push method wherever possible.**

Polling methods, such as a Nexus repository, strain both IBM uDeploy and the repository, which can lead to versions being missed or duplicated.

**Store directory structures matching target machines in component versions.**

This takes advantage of the content optimizations provided by CodeStation, which can save time by deploying only changed artifacts. Nexus, by contrast, stores everything in archives that have to be transferred and expanded before even attempting to detect changes.

**Retrieve deployable artifacts as close to the source as possible.**

If you are deploying non-compiled code, try to get it directly from the source control system. This is especially true for property files that a build system might package in order to fit in a system like Nexus.

**Limit the amount of user input required by processes.**

Set defaults for as many user-supplied values as possible. Try to get values directly from source systems or applications wherever possible. If you have a lot of components, use snapshots.

**During deployment, rely on as few external systems as possible.**

If you connect to an external system from a large number of machines, you incur the overhead of ensuring that all connections and credentials are correct. But if you use IBM uDeploy to store items and values, you know that access is guaranteed when an agent comes online. If you rely on an external password management system, make sure that every agent can connect to it and the credentials are valid.

**Pay attention to possible race conditions.**

IBM uDeploy is a distributed system that allows many concurrent deployments, so make no assumptions about the state of working directories between deployments—anything could have run in them and changed state. Use unique working directories wherever possible. If you need to preserve an artifact, make it part of the component version, or generate it from scratch. Finally, if you suspect multiple components are attempting to access/modify the same file on the same box, use the locking mechanism (step) IBM uDeploy provides.

**Avoid shared drives.**

IBM uDeploy is very efficient at deploying the same copy of a file to multiple machines. Network file systems, by contrast, are dependent on network state and can have issues synchronizing access or dealing with caches, and require you to maintain credentials.

**Tip**

Avoid transferring files between boxes. Rely on IBM uDeploy to keep things synchronized and always use it to transfer files from the server to agents.

**Use sub-resources to represent different entities on the same machine.**

Whenever you have things like web container instances, and directories that are not part of the same application, use sub-resources. This makes inventory management much cleaner and enables you to separate and secure them better.

**Avoid hard-coding values in processes.**

Use properties for everything that might change or differ between environments and machines. Properties are faster and more efficient wherever values are reused.

**Use component templates.**

Templates can establish and enforce standards, and enable faster component creation and modification.

## Maintenance

**Regularly back-up the database and file system.**

The database and file system should be kept in sync.

**Keep track of storage space.**

Component artifacts are stored in the *server install* directory. Based on the number of components, versions, and artifact size the space used by IBM uDeploy will change. If space becomes an issue, adopt a more aggressive clean-up policy (by default, nothing is cleaned up,) or try to reduce the number of new versions.

**Keep track of memory usage.**

Memory usage is primarily affected by the number of concurrent users, including agents running jobs. Spikes in memory usage are usually caused by a large number of deployments, or a large number of users logging in the web user interface. If you start to see spikes, plan on increasing available memory.

**Monitor network usage.**

This enables you to detect deployments in which a lot of machines attempt to download artifacts at the same time, which can lead to slow deployments and potential packet drops. If slow downs occur, add more network capacity or re-design the deployment process to reduce the number of concurrent deployments.

**Stop the IBM uDeploy server during database maintenance.**

The server cannot perform any useful work when the database is off line.

**Except for evaluation, do not run agents on the same machine where the server is located.**

Agents can be resource intensive and cause slow downs for all users.

**Reinstalling agents.**

Preserve a copy of the *conf/agent/installed.properties* file in the *agent\_install* directory or the agent will be treated as a new agent, and you will be warned about using duplicate names. If this happens, you will have to reconfigure all affected applications and resource groups to use the new agent, as well as update its properties and security settings.

**Modifying server and agent locations.**

Server and agent locations are hard coded in their start-up scripts, as is the Java location. If you move or change any of these locations, update the scripts or use soft links.

**Miscellaneous maintenance issues.**

- Logs are automatically rotated.
- Agent upgrades should be performed through the Web user interface
- Make sure the server has actually stopped before performing upgrades or making other changes.
- Always use service scripts to start/stop the server and agents; otherwise, if the account changes, file permissions might prevent them from restarting.

## Rollbacks

IBM uDeploy does not have a built-in roll-back process. Deployment rollbacks are done with compensating actions—actions that undo prior actions. To roll back a deployment, you can re-deploy components using an older snapshot or version. This is possible because IBM uDeploy's artifact repository, CodeStation, maintains all older versions (as long as they have not been cleaned up).

**Rolling back specific steps.**

If you need to run specific steps as part of a roll-back, such as restoring a database, define the steps in a separate component process and execute it as part of an automated roll-back (using the Rollback Component step) in a deployment application process, or keep it in a separate application process.

**Design roll-backs based on production environments.**

In many cases you can roll back a preproduction environment by wiping it clean and recreating the database, which is rarely an option for production environments. If you choose to have environment-based procedures, ensure that they don't inadvertently destroy the production database. Clearly identifying and consistently naming the roll-back processes are good steps, as is limiting the number of people that can run them.

## Security

There are two models to use when setting up security:

- **Enforcement** reduces maintenance by using default permissions to provide broad access rather than defining explicit permissions. Although this leaves the door somewhat open to user abuse, the cost for users is prohibitively high because all transgressions are tracked.
- **Prevention** explicitly defines user access. This model is more secure than *enforcement* but incurs higher maintenance costs. Maintenance overhead can be reduced by using groups.

### Tip

Security Rule #1: Keep the security system as simple as possible. The more complicated the system, the more time required to manage it.

**Use groups.**

Groups make adding and removing users easier, and have the potential to be automatically managed by AD outside of IBM uDeploy.

**Identify a single anchor point and build from there.**

Begin by finding a single level where you can maintain strict security. For example, you might restrict access to deployment requests in upper environments, but let everybody else on the team have full access

to their own applications and components. This method requires diligence from the approvers/requesters to make sure changes have been tested and are non-destructive, but it greatly reduces security maintenance.

### **Use read permissions as top-level access gates.**

If users cannot access something, they cannot do anything with it even if they otherwise have unrestricted access to it. For example, you might grant everybody full administration access to all applications, but control read permissions on a per-team basis. Done this way, teams cannot see each others applications and cannot actually edit or execute them.

### **Define secure properties at the right level.**

If a system-level property holds the production DB password, then any team can use the property and reference it in environments for which it was not intended.

## **Tip**

Properties inherit the security settings of the levels in which they are defined.

### **Protect machines on the agent level.**

Make sure users cannot define a resource as a child of an agent and use it on machines to which they should not have access.

### **Define roles based on actual use.**

By default objects come with administration privileges that provide full access. By adding a read-only user role, you can grant read permissions; and by adding an approver role to an environment, you can easily maintain a single-level approval process for environments.

### **Clearly distinguish roles and groups.**

Roles define access for a specific instance of an object. Avoid using a team or application name as part of a role name. Role names such as admin, owner, user, level 1 approver, config manager, and so forth make sense. Roles names like 'Calypso App Manager,' on the other hand, make better sense as group names.

### **Understand the various security layers.**

An example will illustrate the point: What does a user need in order to deploy to an environment? Execute permissions on: the resources, the environment, the application, and all affected components. If a single layer is missing, the user won't be able to execute a deployment. Generally, you need the same level of access to the parent object (and its parents) before you can get to a specific object.

### **UI and system security.**

While the rest of the security model targets existing objects, UI security controls the visibility of top-level tabs. System security defines who can create applications, components, templates, resources, resource roles, and who can manage plug-ins and overall system security.

### **IBM uDeploy does not cache any authentication data when using AD.**

AD only passes data to AD and does not have access to user passwords. This means you cannot configure processes to use user credentials to run steps on agents.

### **IBM uDeploy security has nothing to do with the accounts used by agents to run processes.**

Accounts are managed by the OS and, unless using impersonation, agent processes run under the account with which the agent was started.

**There is no inheritance in the security model.**

Because you are an administrator for an application does not mean you are an administrator for each of its environments. You have to use the default permissions or explicitly set permissions for each object.



---

# Glossary

## A

Agent	<p>Agents are light-weight Java processes that run on the agent machine. Agents allow the distribution of tasks for performance and multi-platform support. The agent contacts the server whenever the agent process is started. Since the agent communicates with the central server, it need not be on the same network as the central server. However, the agent must be able to open a socket connection to the server. By default, all communication between the central server and the agent is not secure. Communication may be secured using SSL.</p> <p>Agents are light-weight Java processes that run on the agent machine. Agents allow the distribution of tasks for performance and multi-platform support. The agent contacts the server whenever the agent process is started. Since the agent communicates with the central server, it need not be on the same network as the central server. However, the agent must be able to open a socket connection to the server. By default, all communication between the central server and the agent is not secure. Communication may be secured using SSL.</p>
agent	<p>An agent is a lightweight process that runs on a host and communicates with the IBM uDeploy server. Agents manage the resources that are the actual deployment targets.</p>
Agent Filters	<p>Agent Filters are used to select one or more agent(s) at run time and to monitor the quiet period.</p>
agent pools	<p>An agent pool helps you organize and manage agents installed in different environments.</p>
Agent Relays	<p>The new communication relay acts as a proxy for agents (which perform work on behalf of the AnthillPro server) that are located behind a firewall or in another network location. Available under separate download.</p>
agent relays	<p>An agent relay is used to communicate with remote agent. An agent relay requires that only a single machine in the remote network contact the server.</p>
AHPTool	<p>AHPTool is a command-line interface for AnthillPro that provides communication between agent-side commands, scripting and the AnthillPro central server. AHPTool is focused on setting and retrieving information from the AnthillPro server in the context of a running workflow: It can be used to look up or set properties at the system, step, request, job, Build Life and agent levels. AHPTool can upload and retrieve Test, Coverage, Analytics, or Issue data in the form of an XML document to AnthillPro -- making it an excellent integration point for writing your own plug-in or script.</p>
Any Agent Filter	<p>An any agent filter selects the first available agent. Also, it returns all online agents in the environment, ordered by a combination of current load and throughput metric.</p>
applications	<p>An application is the mechanism that initiates component deployment; they bring together components with their deployment target and orchestrates multi-component deployments. An Application must have one component.</p>

application process	An application process can run automatically, manually, or on a user-defined schedule. An application process can orchestrate the entire process including putting servers on-and-off line for load-balancing as required.
Application Security Report	The application security report provides information about user roles and privileges defined for IBM uDeploy-managed applications.
Artifact	For any artifact associated with this project, the files produced by a workflow published as the artifact.
artifacts	Artifacts are files, images, databases, configuration materials, or anything else associated with a software project.
Artifact Retention Policy	Manages how long you will keep the artifacts in order to save disk space.
Artifact Set	An Artifact Set is a label for a collection of build artifacts. Artifact Sets are used to define dependency relationships: Any project that another project depends on generates one or more collections of files used by the dependent project.
Authentication Realm	Authentication Realms are used to determine a user's identity within an Authorization Realm.
Authorization Realm	Authorization realms are used to associate Users with Roles and to determine user access to AnthillPro.

## B

Bar Chart	Displays the data in a bar chart embedded in HTML. Data must all be numeric.
blackout	Blackouts are set per-environment, per-application. Once set, no deployments (nor Snapshots) can be scheduled to occur in that Environment. Any previously scheduled deployments to the Environment will fail if they fall within the blackout date you set.
Build-Farm	The default environment containing all agents used for pure-build processes.
Build Life	represents all the transformations a build has gone through, and the processes such as deployments and testing that the artifacts have undergone.
Build Life Links	Build Life Links are used to pass the URL of resources outside of AnthillPro to the Build Life.
Build Life Originating Workflow Request Property	When set as a workflow property, the value will get pushed to the Build Life originating workflow before the build begins.
Build Life Properties	typically used to hold variable data for builds or to create an audit trail. Once the property is set, and the build has run, the Build Life Property will be visible on the Build Life page.
Build Request Property	Build Life Tools- Build Life Tools are available through the UI to help you diagnose problems and manage Build Lives.
Build Request Property	The property will get pushed to the build request before the build begins.
Build Workflow	A build workflow defines the process for running jobs.

## C

Caching Proxy	Could reduce bandwidth and improve AnthillPro's response time. This is especially helpful in distributed development environments: the proxy can improve performance for users at off-site locations because commonly used pages are loaded from the locally stored cache.
Cleanup	Cleanup configures AnthillPro to periodically cleanup old Build Lives, build request and miscellaneous jobs generated by a project. Cleanup is on a per-project basis, so every project that uses the same Life-Cycle Model will have the same policy. To clean up the Build Life, the user can delete, inactivate, or archive the Build Life.
Cleanup Build Lives	determines when to cleanup build lives. The cleanup of build lives is base on the status the build life has achieved.
Cleanup Policy	Specifies when to delete information about old Build Lives and other task associated with the project.
Cleanup Schedule	A Cleanup Schedule kicks-off the Cleanup.
Cleanup Build Request	A Cleanup Build Request determines when build request are cleaned up.
Clone Instance	Cloning an instance of your AnthillPro server will allow you to change scripts, optimize processes, improve build performance, move the server, etc., without having to experiment on the production server.
CodeStation	CodeStation is the name of AnthillPro's artifact repository management and access tool set. It provides support of dependencies of third-party tool kits and software libraries. CodeStation is also responsible for bringing the dependency management lookup and retrieval utilities to the individual developer.
CodeStation	CodeStation is IBM uDeploy's artifact repository. It provides secure and tamper-proof storage. It tracks artifact versions as they change and maintains an archive for each artifact.
CodeStation Artifact Time-to-Live	This feature determines how long unused artifacts remain in the cache.
components	A component represents deployable items along with user-defined processes that operate on it. Components are deployed to a resource by agents.
component inventory	A component inventory tells you what version of the component is running on a resource.
component process	A component process is a series of user-defined steps that operate on a components artifacts.
Component Security Report	A component security report provides information about user roles and privileges defined for components.
component template	Component templates enable you save and reuse component processes and properties. Components based on templates inherit the template's properties and process.

**Conflict Strategy** A Conflict Strategy allows you to control how AnthillPro acts when a conflict occurs within the dependency tree.

**Cron Schedule** A cron schedule may be configured to consider more complex criteria than a simple interval.

## D

**Defined-value property** Basic property type. Displays a secured field for the user (either the property name or default value) when running a build.

**Dependencies** Specifies the specific artifacts, including version, from other projects that should be retrieved in order to support the workflow. Also specifies the artifacts from other projects and the directory in the checked-out source that those artifacts should be placed in.

**deployment** A deployment is the process of moving software through various preproduction stages to final production.

**Deployment Average Duration Report** A deployment average duration report provides the average deployment time for applications executed during a user-specified reporting period.

**Deployment Count Report** A deployment count report provides information about the number of deployments executed during a user-specified reporting period.

**Deployment Detail Report** The deployment detail report provides information about deployments executed during a user-specified reporting period.

**Deployment Reports** A deployment report provides information about user roles and privileges defined with the IBM uDeploy security system.

**Deployment Total Duration Report** A deployment total duration report provides total deployment for applications executed during a user-specified period.

**design space** The process editor's work area, where plug-in steps are configured and process flows defined.

**Dev-kit** The Dev-kit provides comprehensive documentation on AnthillPro scripting, remoting, using SOAP-based web services with AnthillPro, the AHPTTool (command-line tool) and creating your own Plug-in for AnthillPro.

**digital certificate** A digital certificate is a cryptographically signed document intended to assure others about the identity of the certificate's owner.

**Distributed Servers** Distributed Servers is a complimentary product to AnthillPro designed to reduce the reliance on WAN connections by performing all the heavy lifting (builds, etc.) on local networks. There are three main components to Distributed Servers: Distributed Web Interface, Codestation 2.0, and Agent Relay.

**Distributed Web Interface** The interface mirrors the AnthillPro server, providing the same information and functionality as the main server, for AnthillPro users.

## E

**Empty-value properties** enable users to set a standard for the available attributes a user can provide when executing a resource. Using an empty-value property gives users the option to

	define a required value or simply pass a blank value that AnthillPro will effectively ignore.
environment	An environment is a user-defined collection of one or more resources that host an application. At least one environment must be associated with the process before the process can be executed.
Environment	An environment is a partition grid of agents that is specific for different stages of a project life-cycle.
Environment Group	determines the set of environments that project workflows may be executed on. Each environment group must contain at least one environment.
environment inventory	An environment inventory tells you both what versions of any given component is running on a particular resource.
Environment Property	Environment Properties are used to set default values for a particular environment.
Environment Security Report	An environment security report provides information about user roles and privileges defined for environments.
Event Triggers	Event Triggers use scripts to create an Event Filter that listens to events passing through the AnthillPro Event Service. When the Event Script detects an event, it can then trigger another action by the AnthillPro server.

## F

Fixed Agent Filter	Fixed Agent Filters always select a specific agent within the environment. If the requested agent is locked or can't receive more work, the request will be queued until the agent frees up.
full version	A full version contains all component artifacts.

## G

Guest User Account	The Guest User Account gives anonymous access to AnthillPro, and does not require a user name or password at login.
--------------------	---

## I

IDE Plug-ins	The plug-ins enable developers to view the current activity and state of projects, start new builds, map their projects to projects in AnthillPro, and resolve the project's dependency artifacts.
incremental version	An incremental version contains only artifacts that have been modified since the previous version was created.
Independent Scheduling	Related projects do not need to know about each other for independent scheduling. A dependent project starts each build by gathering the most recent artifacts from its dependencies.
Internal Projects	Allows for dependency relationships between concurrently developed AnthillPro projects.

Interval Schedule                      Regularly fires after a fixed interval of time.

## J

Job    A job is a series of steps detailing how to get something done.

Job-execution Property              Use the Job-execution property type if the value, default value or options are to be generated by a job that executes on an agent.

Job Iteration Property                Job iteration properties are configured after iterating a job. They are typically used to set parameters for a single job that is run many times, with a slightly different parameter each time.

Job Iteration                            Job Iterations are used to run the same job multiple times.

Job Wizard                              A Job Wizard assists in the creation of build jobs. The Wizard takes you through the steps to configure the builder and the publisher required by the job, and is the best way to ensure your build job will be configured successfully. The Build Life is a map of (1) what occurred during the build; (2) the processes that were performed on the generated artifacts; (3) where the build artifacts ended up.

## L

Library Jobs                            A Library Job is a way for you to create template jobs. Once a Library Job has been configured, it can be used by multiple projects or can be turned into a project-specific job.

Library Workflow                      The Library Workflow can only use library jobs as part of its definition.

Life-Cycle Build                        The Life-Cycle Build provides a wealth of information and visibility into the build and release cycle. It is defined by its Life-Cycle Model, Environments, and Build workflow.

Life-Cycle Model                        The Life-Cycle Model provides a template for managing the dependencies, artifacts, deployments, etc, associated with every build of the project. They are reusable, allowing you to apply the same standards to any similar project.

LDAP                                      A lightweight directory access protocol (LDAP) is a widely used protocol used to access distributed directory information over the internet protocol (I.P) networks.

lock                                        A lock is routinely used to ensure that processes do not interfere with one another.

Lockable Resource                      A Lockable Resource specifies a resource that might be used by multiple workflows, gives it a name and forces arbitrary workflows to run one at a time or in small groups.

## N

Network Settings                        Network Settings are used to configure communication between the AnthillPro server and agents.

Notification                            Notifications inform the recipient of the status of a CI build.

notifications                            Notifications play a role in approving deployments: IBM uDeploy can be configured to send out an e-mail to either a single individual or to a group of people

(based on their security role) notifying them that they need to approve a requested deployment.

**Notification Scheme** Notification Schemes determine who to notify of build status, what conditions to notify them on, and what mechanism to notify them with. sets rules determining what groups of users are sent which kind of notification about specific events.

**notification scheme** A notification scheme enables IBM uDeploy to send out notifications based on events. For example, the Default Notification scheme will send out an e-mail when an Application Deployment fails or succeeds.

**Notification Template** Notification Templates are Velocity templates that take information about the build and produce a document.

## O

**Operational Project** Operational Projects provide a simplified interface that allows AnthillPro users to execute ancillary tasks not easily run during a build, deployment, etc., workflow.

## P

**Permission** Permissions associate the role, resource, and an action that may be performed on the resource. Typical actions include the ability to read or view the resource, the ability to write to or modify the resource, the ability to modify the security settings for a resource, or the ability to execute the resource.

**plug-ins** A plug-in is the integration with third-party tools.

**Preflight Build** Preflight Builds allow developers to run a test build in the authoritative build environment before committing their changes to source control.

**process** Processes play a coordination role. They are authored using a visual drag-n-drop editor.

**process editor** A process editor is a visual drag-and-drop editor that enables you to drag process steps onto the design space and configure them as you go.

**processing property** A processing property is a way to add user-supplied information to a process.

**Project Environment Property** Project Environment Properties are automatically placed as environment variables for all commands run in the target environment.

**Project Property** Project Properties are used for all workflows regardless of the target environment.

**Property** Properties identify various properties that must be specified when the workflow is executed. Properties are used to manage variables passed into commands, agent filters, and custom stamping algorithm templates.

**proxy resource** A proxy resource is a resource effected by an agent on a host other than the one where the resource is located.

**Pull Build** A Pull Build will build every dependency that is out of date prior to building the top level of a project.

**Pure Build** Pure builds take the source as input and transforms it into artifacts. In AnthillPro, it generates the Build Life.

**Push Build** Push Builds perform the minimum number of builds, in the correct order, to ensure that a change to a component does not break anything that depends on it directly or transitively. A push build can become resource intensive.

**Push Scheduling** Push Scheduling are used when an originating workflow builds, the workflows that depend on it automatically build.

## Q

**Quiet Periods** Quiet Periods are configured on the project, and play an integral part in ensuring that the source code AnthillPro obtains from the SCM contains a consistent set of changes. The quiet period is a measure of time that must pass without anyone committing changes to the source.

## R

**Read Permission** Read permissions allow a user to resolve/download the artifacts associated with a Build Life.

**relay servers** A relay server enables network-to-network communication.

**remote agents** A remote agent is an agent that will communicate with the server via an agent relay.

**Report** Reports provide information about system activity.

**Repository Trigger** Repository Triggers are used for Continuous Integration builds. Once the trigger is active for a workflow, every commit of source changes will initiate a build.

**resource** A resource is a user-defined construct based on IBM uDeploy's architectural model. A resource represents a deployment target.

**resource group** A resource group is a group of resources used to help organize and manage the agent installed in a different environment.

**Resource Security Report** A resource security report provides information about user roles and privileges defined for resources.

**role** A role enables you to further refine how a resource is utilized, and is similar to subresources.

## S

**Schedule** A Schedules determines when events such as builds, cleanups, backups, etc., are automatically run by the system. Once a schedule has been created, it may be used by many different AnthillPro resources.

**Schedule Trigger** A Scheduled Trigger runs on its own timer that pools the SCM for changes. If changes are detected, the build is registered with the schedule and kicked off when the schedule fires. Scheduled Triggers give the option to force a build regardless of whether source changes have occurred.

**schema** A schema is a visual representation of the different parts of IBM uDeploy that may be secured. Each Schema interacts with users indirectly, through the role.



Script Library	the script library is used to create, organize, and provide security around often-used AnthillPro scripts. The Script Library is most helpful for large organizations, allowing them to ensure that only the appropriate team members can modify a script.
Scripted Property	Scripted value properties are properties where the value, default value or options are generated by a script that is executed before workflow execution.
Scripted Agent Filters	A scripted agent filter selects agent based on an agent filter script.
SSL	A secure socket layer (SSL) enables clients and servers to communicate securely by encrypting all communications.
Security Permission	Security Permissions allows users to determine who can set security for the artifact set.
Security Reports	A security report provides information about user roles and privileges.
Security Setting	A Security Setting is used to configure access to the Anthill Server setting for configuration and artifact management.
Server Proxy	Server proxies enable you to access servers/repositories that are on external networks.
snapshot	A snapshot is a collection of specific component versions, usually versions that are known to work together.
Source Configuration	Source Configuration identifies exactly which source code should be retrieved from a repository.
Stamp	A Stamp is a configurable name for the build number, build identifier, or version number used to identify a build. This allows you to mimic your strategies.
Stamp Context Script	Stamp Context Scripts generate values for variables in the stamp context when creating a stamp (for builds, etc.).
Stamp Mapping	For every stamp style in the project's stamp style group, one must specify what stamping strategy will be used. A typical stamp style group might specify for development builds to a version number and strategy for incrementing that number.
Stamp Style	Stamp Styles are used to apply different stamps to a single build, and allow you to help track a build throughout its life-cycle.
Stamp Style Group	A Stamp Style Group creates common names for types of stamps.
Status Group	A Status Group is a set of common names for statuses.
stateless	Stateless means the server retains little session information between requests, and each request contains all information needed to handle it.
subresource	A subresource enables you to apply logical identifiers or categories within any given group.
switch step	A switch step enables you to create conditional processes.
System Tray Monitor	The System Tray Monitor provides feedback directly to the desktop, without having to open or refresh a browser.

**System Property** The System property is used to set default values for a particular property for all workflows and project system wide.

## T

**Template Reports** Template Reports are good for generating tabular data in one or more types of output.

**Trigger** A Trigger is an automated mechanism for kicking off a workflow.

## U

**uncontrolled environment** A uncontrolled environment is an environment that does not contain approvals approval gates.

**user impersonation** IBM uDeploy can use user impersonation when an agent must execute a command for which it might not otherwise have permission.

## V

**Velocity Report** Velocity Reports are good for customizing the AnthillPro UI.

**version** A version is set each time a component changes. There are two types of versions a full version and an incremental version.

## W

**Workflow Definition** A Workflow Definition defines which job should be run, species the order of jobs, as well as the elements to be run in parallel.

**Workflow Priorities** Workflow Priorities allow you to determine the order in which workflows run. Use workflow priorities to determine which workflow will run first.

**Workflow Property** Workflow properties are used to send a build to a particular platform when writing native code for multiple platforms.

**Workflow Request Context** A workflow request context is a collection of requests for workflows that are processed together.

**Workflow Task** A Workflow Task allow you to set up manual gates, etc., that must be performed.

**Workflow Tool** There are two major categories of work flow tools: priorities and requests.

**Workflow Request** The Workflow Request is the first action taken by the server when executing an originating or secondary workflow.

---

# Index

## Symbols

"/" character in step names, 121  
\${p:application/propertyName}, 132  
\${p:environment/propertyName}, 82, 132  
\${p:propertyName}, 133  
\${p:stepName/propName}, 125, 131  
\${p:system/propertyName}, 133  
\${p:version.name}, 132

## A

access gate, 199  
Activate action, 52  
active inventory status, 54  
active status, 85, 85, 86  
addAgentToPool, 139  
adding components to applications, 78  
adding environments to applications, 80  
agent, 72  
    agent pools, 75  
    installing, 16  
    remote agents, 73  
agent names, 197  
agent pools, 75  
agent security, 199, 199  
agent.id, 133  
agent.name, 134  
anchor point, 35  
application process  
    manual task, 87  
application process steps  
    Finish, 84  
    Install Component, 84  
    Manual Application Task, 87  
    Rollback Component, 86  
    Uninstall Component, 85  
application properties, 132  
application role, 83  
application.id, 133  
application.name, 133  
ApplicationDeploymentFailure, 62, 87  
ApplicationDeploymentSuccess, 62, 87  
applications, 76  
    add environment, 80  
    adding components, 78  
    creating, 77  
    creating processes, 83  
    exporting, 78  
    importing, 80  
    manual task, 87  
    mapping resources, 81

    offline agent, 84  
    process steps, 84  
    processes, 82  
    role, 83

Approval Failed, 62, 87  
ApprovalCreated, 62, 87

## B

backing up, 197  
best practices, 195  
blackout, 96

## C

clean-up policy, 197  
CLI, 136  
CLI command format, 136  
command element (plug-in), 122  
command line interface (CLI), 136  
component process, 52  
    running, 54  
    type, 54  
component process steps  
    Manual Task, 62  
component processes  
    manual task, 62  
component role, 54  
component version  
    auto, 51  
    creating, 50  
    deleting, 52  
    full or incremental, 50  
    inactivating, 52  
    status, 52  
component.id, 133  
component.name, 133  
components  
    adding to applications, 78  
    manual task, 62  
    post-processing, 63  
    process type, 53  
    processes, 52  
    role, 54  
    versions, 49  
connection tool, 59  
content optimization, 196  
createAgentPool, 148  
createResource, 152  
creating application processes, 83  
creating applications, 77  
creating plug-ins, 118, 125

## D

deleteAgent, 157

- deleteAgentPool, 157
- deleteResource, 158
- deleting component version, 52
- Deployment Detail report, 98
- deployment reports, 97
  - Deployment Detail, 98
- DeploymentReadied, 62, 87
- duplicate agent names, 197

## E

- enforce complete snapshots, 77
- environment properties, 82, 132
- environment.id, 133
- environment.name, 133
- environments, 80
  - adding to applications, 80
  - mapping resources, 81
- escaped property characters, 135
- exitCode, 62, 134
- exporting applications, 78

## F

- file system basic, 115
- file system versioned, 115
- Finish process step (application), 84
- full component version, 50

## G

- generic processes, 92
- getAgent, 159
- getAgentPool, 160
- getAgentPools, 160
- getAgents, 160
- getApplicationProperties, 162
- getApplicationProperty, 162
- getComponentEnvironmentProperties, 163
- getComponentEnvironmentProperty, 163
- getComponentProperties, 165
- getComponentProperty, 165
- getEnvironmentProperties, 166
- getEnvironmentProperty, 166
- getEnvironmentsInApplication, 167
- getGroupsForResource, 167
- getMappingsForApplicationEnvironment, 168
- getMappingsForGroup, 168
- getResourceProperties, 170
- getResourceSecurity, 171
- getRolesForResource, 173
- getSystemProperties, 173
- getSystemProperty, 174
- global properties, 133
- groups, 198

## I

- importing applications, 80
- inactivateEnvironment, 175
- inactivating component version, 52
- incremental component version, 50
- incremental deployments, 195, 195
- info.xml, 125
- Install Component process step (application), 84
- installAgent, 176
- installing agents, 16
- installing plug-ins, 119

## J

- Java home path, 73
- java.util.Properties, 123, 131

## K

- keystore, 22, 22
- keytool, 22

## L

- lines of interest, 62, 124, 134
- LOCAL SYSTEM account, 17
- Lock Snapshots check box, 37
- locking snapshots, 80
- LOI, 62, 134

## M

- maintenance, 197
- Manual Application Task process step (application), 87
- manual task (application), 87
- manual task (component), 62
- Manual Task component process step, 62
- mapping resources to an environment, 81
- memory usage, 197
- modifying agent locations, 197
- mutual authentication, 22

## N

- network usage, 197
- Nexus, 196
- notification scheme, 77

## O

- offline agent handling, 84
- Oracle
  - installing, 11
  - supported editions, 11

## P

- plug-in
  - command element, 122

- creating, 118
- example, 125
- info.xml, 125
- installing, 119
- plugin.xml, 119, 129
- post-processing element, 123, 131
- step name, 121
- step-type, 121
- upgrading, 125
- plug-in command element, 122
- plug-in step structure, 126
- plug-in step-type, 121
- plugin.xml, 119, 129
- polling, 196
- post-processes, 63
- post-processing element, 123, 131
- post-processing properties, 125, 131
- precondition, 85, 86, 86
- process properties, 133
- process steps (application), 84
- ProcessRequestStarted, 62, 87
- properties
  - `{p:version.name}`, 132
  - escaped characters, 135
  - format, 132
- property format, 132
- property security, 199
- proxy host, 74
- push method, 196

## Q

- quick start
  - applications, 5
  - deployments, 5

## R

- race conditions, 196
- remote agent, 73
- removeAgentFromPool, 179
- removeMapping, 182
- removeVersionStatus, 188
- resource.id, 133
- resource.name, 133
- resources
  - agent pools, 75
  - agents, 72
  - mapping to an environment, 81
  - remote agents, 73
- roadmap, 4
- role (applications), 83
- role (component), 54
- Rollback Component process step, 86
- Rollback Component step, 198

- rollback source, 86
- rollbacks, 198
- rolling deployment, 30, 54
- running component processes, 54
- running generic processes, 92

## S

- scanner, 124
- scanner.scan();, 125, 131
- secure socket layer, 21
- security best practices, 198
- security groups, 198
- security inheritance, 200
- security reports, 108
- Serena Dimensions, 116
- server
  - user account, 8
- setSystemProperty, 192
- Show Inactive Versions, 52
- shutdownAgent, 193
- snapshots
  - locking, 80
- source configuration, 114
  - file system basic, 115
  - file system versioned, 115
  - Serena Dimensions, 116
- SSL configuration, 20
- staged inventory status, 54
- staged status, 85, 85, 86
- standard out, 13
- step name forbidden character, 121
- step-type, 126
- storage space, 197
- sub-resources, 197

## T

- task (application), 87
- task (component), 62
- TaskCreated, 62, 87
- Template Name field, 62, 87
- templates, 197
- testAgent, 193

## U

- udclient, 136
- Uninstall Component process step (application), 85
- upgrade.xml, 118
- upgrading, 20
- upgrading agents, 20
- upgrading plug-ins, 125

## V

- version (component), 49

version.ID, 133  
version.name, 133