



SERENA®

RELEASE AUTOMATION

User's Guide

Serena Proprietary and Confidential Information

Copyright © 2011–2013 Serena Software, Inc. All rights reserved.

This document, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by such license, no part of this publication may be reproduced, photocopied, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Serena. Any reproduction of such software product user documentation, regardless of whether the documentation is reproduced in whole or in part, must be accompanied by this copyright statement in its entirety, without modification. This document contains proprietary and confidential information, and no reproduction or dissemination of any information contained herein is allowed without the express permission of Serena Software.

Trademarks

Serena, TeamTrack, StarTool, PVCS, Comparex, Dimensions, Prototype Composer, Mariner and ChangeMan are registered trademarks of Serena Software, Inc. The Serena logo, Version Manager and Mover are trademarks of Serena Software, Inc. All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

U.S. Government Rights

Any Software product acquired by Licensee under this Agreement for or on behalf of the U.S. Government, its agencies and instrumentalities is "commercial software" as defined by the FAR. Use, duplication, and disclosure by the U.S. Government is subject to the restrictions set forth in the license under which the Software was acquired. The manufacturer is Serena Software, Inc., 1850 Gateway Drive, 4th Floor, San Mateo, California, 94404-4061.

Product version: 4.0.1

Publication date: 2013-03-07

Table of Contents

Serena Release Automation 4.0.1	1
About This Documentation	9
Organization of this Documentation	9
Product Support.....	9
Documentation Conventions	10
Overview	10
Components	12
Component Processes	12
Application Process	14
Environments	14
Snapshots	14
Plug-ins	15
Component Versions and the CodeStation Repository	15
Applications	15
Snapshots	15
Agents	15
Resources	16
Resource Groups	16
Architecture.....	16
Service Tier.....	17
Clients	19
Relational Database	19
File Storage—Codestation	20
Data Center Configuration	20
Cold Standby	20
Platform Considerations	20
Typical Data Center Configurations	21
Agents	23
Server-Agent Communication.....	24
Remote Agents: -Crossing Network Boundaries and Firewalls	25
Agent Security	26
User Impersonation	26
Using su/sudo	26
Impersonation on Windows Systems.....	27
SSL Mutual Key-based Authentication	27
Getting Started	29
Serena Release Automation Roadmap	29
Installing Serena Release Automation	29
Creating a Component.....	30
Creating an Application	30
Deploy the Component	31
Installing Servers and Agents.....	32
Installation Recommendations	32
System Requirements	33
Server Minimum Installation Requirements	33
Server Installation Recommendations	33
Agent Minimum Requirements	34
JVM Support (32- and 64-bit).....	34
Downloading Serena Release Automation	34
Database Installation	35
Installing Oracle.....	35
Installing MySQL	36
Installing Microsoft SQL Server	36
Server Installation	37
Interactive Server Installation (Windows, Linux/UNIX (AIX, Solaris)	38
Server Installation (Other UNIX Platforms).....	38
Silent Mode Server Installation	38
Server Silent Install (Windows)	39
(Windows) Server Silent Install Options	39

(Windows) DB Examples - Server Silent Install	39
Server Silent Install (Linux/UNIX (AIX, Solaris))	41
(Linux/UNIX (AIX, Solaris)) Server Silent Install Options	41
(Linux/UNIX (AIX, Solaris)) DB Examples - Server Silent Install	42
Agent Installation	43
Interactive Agent Installation (Windows, Linux/UNIX (AIX, Solaris))	44
Silent Mode Agent Installation	44
Agent Silent Install (Windows)	44
Installing Agent Relays	44
Upgrading Serena Release Automation	46
SSL Configuration	46
Configuring SSL Unauthenticated Mode for HTTP Communications	47
Configuring Mutual Authentication	47
Running Serena Release Automation	49
Running the Server	49
Running an Agent	49
Running an Agent Relay	49
Accessing Serena Release Automation	49
Quick Start—helloWorld Deployment	50
helloWorld: Creating Components	50
helloWorld: Deployment	51
helloWorld: A Note Before You Begin	51
helloWorld: Component Version	51
helloWorld: Component Process	53
helloWorld: Process Design	54
helloWorld: Application	59
helloWorld: Creating an Application	60
helloWorld: Adding the helloWorld Component to the Application	60
helloWorld: Adding an Environment to the Application	60
helloWorld: Adding a Process to the Application	62
helloWorld: Designing the Process Steps	62
helloWorld: Running the Application	64
Using Serena Release Automation.....	67
Components	67
Creating Components	67
Importing/Exporting Components	69
Exporting Components	69
Importing Components	69
Component Properties	70
Component Versions	71
Importing Versions Manually	71
Importing Versions Automatically	73
Component Version Statuses	73
Deleting Component Versions.....	74
Inactivating Component Versions	74
Component Change Logs	74
Component Processes	74
Configuring Component Processes	74
Process Editor	75
Displaying the Process Editor	75
Using the Process Editor	77
Adding Process Steps	78
Connecting Process Steps	80
Process Properties	82
Switch Steps and Conditional Processes	83
Component Manual Tasks	85
Creating Component Manual Tasks	85
Using Component Manual Tasks	86
Post-Processes	86
Component Templates	86
Creating a Component Template	86
Importing/Exporting Templates	87
Exporting Templates	87
Importing Templates	87

Component Template Properties	88
Using Component Templates	89
Configuration Templates	90
Deleting and Deactivating Components	90
Resources	90
Resource Groups	91
Creating a Resource Group	91
Resource Roles	92
Role Properties	93
Agents	93
Remote Agent Installation	93
Managing Agents Remotely	94
Agent Pools.....	95
Creating an Agent Pool.....	95
Managing Agent Pools	95
Applications.....	95
Application Environments	96
Application Processes	96
Snapshots	96
Creating Applications	97
Adding Components to an Application	97
Importing/Exporting Applications	98
Exporting Applications	98
Importing Applications	99
Creating an Environment	100
Mapping Resources to an Environment	100
Environment Properties	101
Creating Application Processes	102
Application Process Steps	102
Application Process Steps Details	102
Finish.....	103
Install Component	103
Uninstall Component	103
Rollback Component	104
Manual Application Task (Utility)	104
Application Manual Tasks	105
Creating Application Manual Tasks	105
Using Manual Tasks	105
Approval Process	106
Work Items	106
Creating Snapshots	106
Application Gates	107
Creating Gates	107
Structure of the default.xml File	109
Deployments	110
Scheduling Deployments.....	113
Reports.....	114
Deployment Reports.....	114
Deployment Detail Report	114
Deployment Detail Fields	115
Running the Deployment Detail Report.....	115
Report Samples: Deployment Detail	116
Deployment Count Report	117
Deployment Count Fields	117
Running the Deployment Count Report	117
Report Samples: Deployment Count	119
Deployment Average Duration Report	120
Deployment Average Duration Fields	120
Running the Deployment Average Duration Report	120
Sample Reports: Deployment Average Duration	122
Deployment Total Duration Report	122
Deployment Total Duration Fields	122
Running the Deployment Total Duration Report.....	123
Sample Reports: Deployment Total Duration	124

Security Reports	124
Application Security Report	125
Application Security Fields	125
Component Security Report	125
Component Security Fields	125
Environment Security Report	125
Environment Security Fields	126
Resource Security Report	126
Resource Security Fields	126
Saving and Printing Reports	126
Saving Report Data	126
Saving Report Filters	127
Printing Reports	127
Administration	128
Serena Release Automation Security	128
Setting up Security	128
Roles and Permissions	129
Default Roles	129
Creating and Editing Roles	130
Agent Roles	131
Application Roles	131
Component Template Roles	132
Component Roles	132
Environment Roles	132
License Roles	133
Resource Roles	133
Default Permissions	133
Setting Default Permissions	133
Authorization Realms	134
Creating an LDAP Authorization Realm	135
Creating Authorization Groups	135
Authentication Realms	136
Creating an Authentication Realm	136
Creating an LDAP Authentication Realm	137
Authentication Realm User	137
Importing LDAP Users	137
To Import LDAP Users	138
Tokens	138
User Interface Security	138
System Security	139
System Settings	139
Licenses	140
Adding a License	140
Adding Agents to a License	140
Modifying or Deleting a License	141
Logging	141
Network Relay	141
Notifications	142
Creating Notification Templates	144
Post-Processing Scripts	144
Preview Version Cleanup	145
Output Log	145
Locks	145
Managing Locks	145
Installing Plug-ins	146
Configuration	146
Application Configuration	147
Adding Application Configuration Properties	148
Modifying and Deleting Application Configuration Properties	149
Component Configuration	150
Environment Configuration	150
Inventory	150
Resources Inventory	151
Component Inventory	151

Environment Inventory	152
Reference.....	153
Basic Fields	153
File System (Basic)	154
File System (Versioned)	154
Serena Dimensions CM	155
Plug-ins	155
Standard Plug-ins	158
Creating Plug-ins	158
- plugin.xml File	159
<header> Element.....	160
Plug-in Steps: step-type Element	160
Step Properties: properties Element	161
<command> Element	162
<post-processing> Element	163
Upgrading Plug-ins	164
The info.xml File.....	164
iSerena Release Automation Properties	164
Command Line Client (CLI).....	166
Command Format	166
Commands	166
addActionToRoleForApplications.....	166
addActionToRoleForComponents	167
addActionToRoleForEnvironments	167
addActionToRoleForResources	168
addActionToRoleForUI	168
addComponentToApplication	168
addGroupToRoleForApplication	169
addGroupToRoleForComponent	169
addGroupToRoleForEnvironment	170
addGroupToRoleForResource	170
addGroupToRoleForUI	171
addLicense	171
addNameConditionToGroup.....	171
addPropertyConditionToGroup	172
addResourceToGroup	172
addRoleToResource	173
addRoleToResourceWithProperties	173
addUserToGroup	174
addUserToRoleForApplication	174
addUserToRoleForComponent.....	174
addUserToRoleForEnvironment	175
addUserToRoleForResource	175
addUserToRoleForUI	176
addVersionFiles	176
addVersionStatus	177
createApplication	177
createApplicationProcess	178
createComponent	178
createComponentProcess.....	179
createDynamicResourceGroup	179
createEnvironment	179
createGroup	180
createMapping	180
createResourceGroup	181
createRoleForApplications	181
createRoleForComponents	181
createRoleForEnvironments	182
createRoleForResources	182
createRoleForUI	182
createSubresource.....	183
createUser	183
createVersion	184
deleteGroup.....	184

deleteResourceGroup	184
deleteResourceProperty	185
deleteUser	185
exportGroup	185
getApplication	186
getApplicationProcess	186
getApplicationProcessRequestStatus	186
getApplications	187
getComponent	187
getComponentProcess	187
getComponents	188
getComponentsInApplication	188
getEnvironment	188
getEnvironmentsInApplication	189
getMapping	189
getResource	189
getResourceGroup	190
getResourceGroups	190
getResourcesInGroup	190
getResources	191
getResourceProperty	191
getRoleForApplications	191
getRoleForComponents	192
getRoleForEnvironments	192
getRoleForResources	192
getRoleForUI	193
getUser	193
importGroup	193
importVersions	194
login	194
logout	194
removeActionFromRoleForApplications	195
removeActionFromRoleForComponents	195
removeActionFromRoleForEnvironments	196
removeActionFromRoleForResources	196
removeActionFromRoleForUI	196
removeGroupFromRoleForApplication	197
removeGroupFromRoleForComponent	197
removeGroupFromRoleForEnvironment	198
removeGroupFromRoleForResource	198
removeGroupFromRoleForUI	198
removeResourceFromGroup	199
removeRoleForApplications	199
removeRoleForComponents	200
removeRoleForEnvironments	200
removeRoleForResources	200
removeRoleForUI	201
removeRoleFromResource	201
removeUserFromGroup	201
removeUserFromRoleForApplication	202
removeUserFromRoleForComponent	202
removeUserFromRoleForEnvironment	202
removeUserFromRoleForResource	203
removeUserFromRoleForUI	203
repeatApplicationProcessRequest	204
requestApplicationProcess	204
setComponentEnvironmentProperty	204
setComponentProperty	205
setResourceProperty	205
updateUser	206

Serena Release Automation 4.0.1

About This Documentation

This documentation guides you through installing and using the Serena Release Automation product and is intended for all users.

This documentation is available in PDF and HTML formats:

- The HTML Help is located online at: http://help.serena.com/oalm_help/4_0_1/sra/index.html.
- The HTML Help can also be accessed from within the product's web-based, user interface by clicking the Help button.
- The pdf is located on the Serena Documentation portal at: <http://support.serena.com/Case/CaseHome.aspx>

You will need to log in using your customer account. If you do not have an account, please contact your Serena Sales Representative.

Organization of this Documentation

The flow of this documentation covers the following high-level areas.

Section	Description
Overview [page 10]	Provides an overview of the product's significant features and describes its architecture.
Getting Started [page 29]	Provides a roadmap to Serena Release Automation productivity, describes how to install the product, and contains a step-by-step introductory tutorial.
Using Serena Release Automation [page 67]	Contains comprehensive chapters for Serena Release Automation's core features, such as components, applications, and resources.
Administration [page 128]	Describes Serena Release Automation's security system and explains how to configure product features.
Reference	Contains several reference-type chapters on topics like: the command-line interface, product properties, writing plug-ins, as well as others.

Product Support

The Serena Support portal, <http://support.serena.com/Case/CaseHome.aspx>, 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

NOTE

You need an account to log in to the Serena Support portal. If you do not have an account, contact your Sales Representative.

Documentation 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 Serena Release Automation 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.

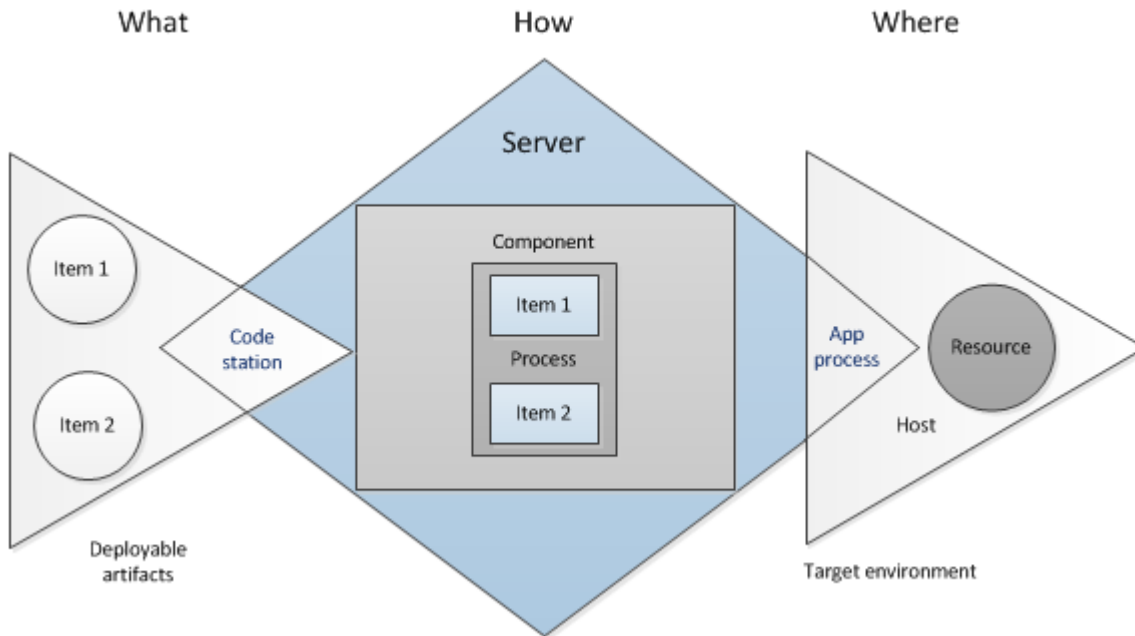
- User interface objects, such as field and button names, are displayed with initial Capital Letters.
- Variable text in path names or user interface objects is displayed in *italic* text.
- Information you are supposed to enter is displayed in this format.

Overview

At its base, software deployment is a simple concept that sometimes gets obscured by jargon. A deployment is the process of moving software (broadly defined) through various preproduction stages to final production. Typically, each stage represents a step of higher criticality, such as quality assurance to production. Complexity arises from the sheer volume of things deployed, the number and variety of deployment targets, constantly-decreasing deployment cycles, and the ever-increasing rate of technological change. While virtualization provides some relief to the process, it also—perhaps paradoxically—increases the challenge with its exponential growth of deployment targets.

Serena Release Automation helps you meet the challenge by providing tools that improve deployment speeds while simultaneously improving their reliability. Serena Release Automation's release automation tools provide complete visibility into *n*-tiered deployments, enabling you to model processes that orchestrate complex deployments across every environment and approval gate. Serena Release Automation's drag-and-drop design tools decrease design-time by making it easy to visualize the end-to-end deployment process and develop the big picture—the *What*, *How*, and *Where* of the deployment workflow:

- **What:** the deployable items—binaries, static content, middleware updates, database changes and configurations, and anything else associated with the software—that Serena Release Automation delivers to target destinations.
- **How:** refers to combining deployable items with processes to create components, and designing applications that coordinate and orchestrate multi-component deployments.
- **Where:** the target destination's hosts and environments—Serena Release Automation can scale to any environment.



In Serena Release Automation, deployable items are combined into logical groupings called components. Components are deployed by component processes which consist of user-configured steps, many taken from integrations with third-party tools called plug-ins. Multi-component deployments are handled by user-assembled *applications*.

Serena Release Automation represents deployment targets by what it calls *resources*. Resources—databases, servers, and so on—reside on hosts. Complex deployments can contain numerous components that target multiple hosts. Deployments are managed by agents residing on the hosts. Components can also remain independent of one another, which enables incremental or targeted deployments. Of course, you can model your components as you see fit—Serena Release Automation is flexible and works the way you work.

Server

The Serena Release Automation server is a standalone server that provides Serena Release Automation's core services such as the user interface, component and application configuration tools, workflow engine, and security services, among others. Many services are REST-based.

Serena Release Automation supports cross-network deployments with relay servers. Relay servers enable network-to-network communications.

Agents

An agent is a lightweight process that runs on a host and communicates with the Serena Release Automation server. Agents manage the resources that are the actual deployment targets. Each machine participating in a deployment usually has an agent installed on it. When not performing deployments, agents run in the background with minimal overhead (see [Resources \[page 16\]](#)).

Repository

The Serena Release Automation-supplied artifact repository, CodeStation, provides secure and tamper-proof storage. It tracks artifact versions as they change and maintains an archive for each artifact. Associations between repository files and components are built-in and automatic.

Security

In Serena Release Automation's role-based security system, users are assigned roles, and role-permissions are assigned to things such as projects, build configurations, and other resources. For example, a developer may be permitted to build a project, but only view non-project related material.

Components

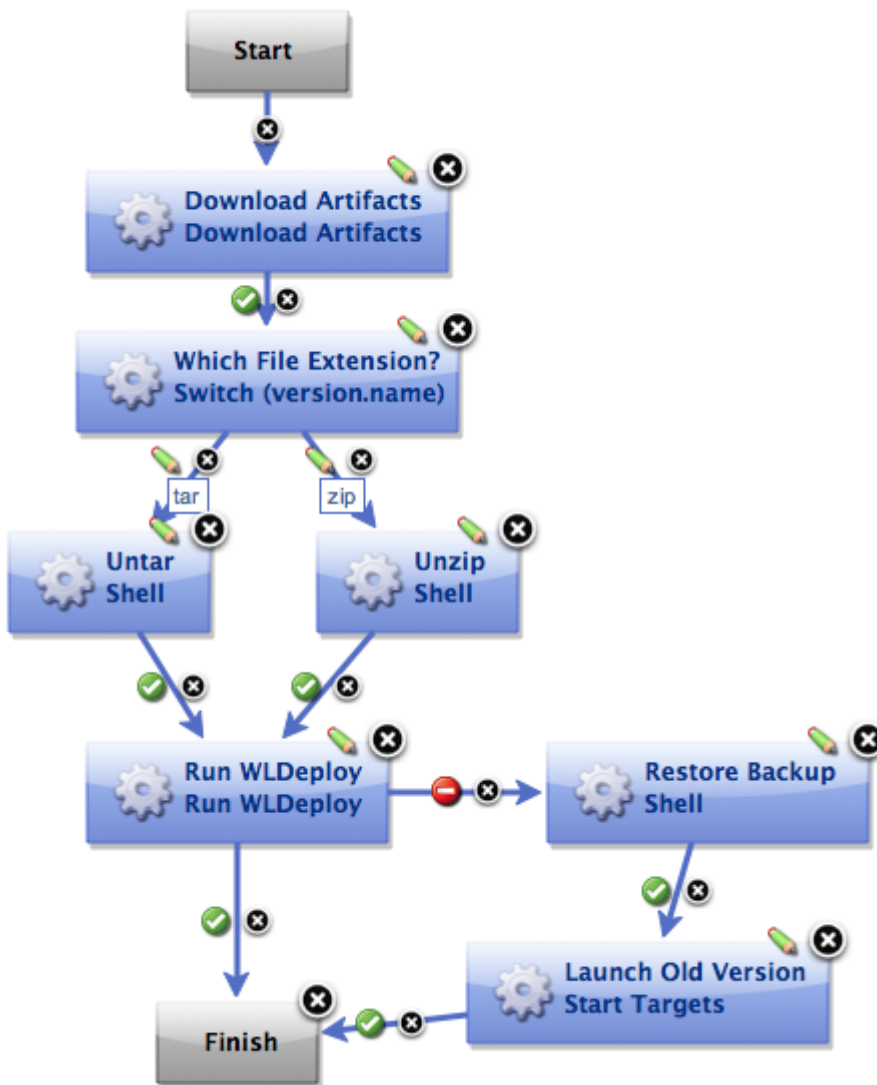
Understanding how Serena Release Automation uses the term component is critical to understanding Serena Release Automation. Components represent deployable items along with user-defined processes that operate on them, usually by deploying them. Deployable items—also called artifacts—can be files, images, databases, configuration materials, or anything else associated with a software project. Components have *versions* which are used to ensure that proper component instances get deployed.

Artifacts can come from a number of sources: file systems, build servers such as AnthillPro, source version control systems, Maven repositories, as well as many others. When you create a component, you identify the source and define how the artifacts will be brought into Serena Release Automation. If the source is Subversion, for example, you specify the Subversion repository containing the artifacts. Each component represents artifacts from a single source.

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 steps and relationships; the switch step enables you to create conditional processes. For example, you might take artifacts from a source—such as an AnthillPro project—and map the ones that get deployed to an HTTP server into one component; those that get deployed to a J2EE container to another; and those that get deployed to a database to yet another. Another example, a single-component deployment might consist of two processes: the first moves component files to a server on Friday night (a lengthy operation), while the second deploys the files Saturday morning.



Component processes are created with Serena Release Automation's process editor. The *process editor* is visual drag-and-drop editor that enables you to drag process steps onto the *design space* and configure them immediately.

As you place additional steps, you visually define their relationships with one another. Process steps are selected from a menu of standard steps that replace typical deployment scripts and manual processes.

Serena Release Automation provides steps for several utility processes, such as inventory management, and workflow control. Additional process steps are provided by plug-ins. *Plug-ins* provide integration with common deployment tools and application servers, such as WebSphere, Microsoft IIS, and many others.

Out-of-the-box, Serena Release Automation provides plug-ins for many common processes, such as downloading and uploading artifacts, and retrieving environment information. A component process can have steps from more than one plug-in. A component process is defined for a specific component. A component can have more than one process defined for it, but each component requires at least one process.

For example, deploying a J2EE EAR file to WebSphere server typically consists of the following operations:

1. Transfer the EAR file to the target machine.
2. Stop the WebSphere server instance.
3. Invoke wsAdmin with deployment properties.
4. Start the WebSphere instance.

5. Verify that the deployment succeeded by accessing a specified URL.

NOTE

The WebSphere plug-in provides a configurable process step for each operation.

A frequently used component process can be saved as a template and applied later to new components.

Component processes are executed by Serena Release Automation agents running on hosts. One instance of a component process is invoked for each resource mapped to a component in the target environment.

Application Process

When you create an application, you identify the included components and define an application process. Application processes, like component processes, are created with the process editor. Serena Release Automation 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. For instance, an n -tiered application might have a web tier, a middleware tier, and a database tier. And, continuing the example, the database tier must be updated before the other two, which are then deployed concurrently. An application can orchestrate the entire process, including putting servers on- and off-line for load-balancing as required.

When an application process executes, it interacts with a specific environment. An environment is a collection of one or more resources. 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. This enables a single application to interact with separate environments, such as QA, or production. To use the same application 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. Serena Release Automation tracks the history of each component version, which enables application processes to restore environments to any desired point.

Environments

An environment is a user-defined collection of resources that host an application. 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.

Serena Release Automation maintains an inventory of every artifact deployed to each environment and tracks the differences between them.

Snapshots

A snapshot is a collection of specific component versions, usually versions that are known to work together. Typically, a snapshot is generated in an uncontrolled environment—meaning one without— approvals. When a snapshot is created, a picture of the application's current state is captured. As an application moves through different environments, snapshots can ensure that proper component versions are used.

Snapshots help manage complex deployments—deployments with multiple tiers and development teams. For example, after testing and confirming that team A's component works with teams B's, a snapshot can be taken. Then, as development progresses, additional snapshots can be taken and used to model the effort and drive the entire deployment, coordinating versions, configurations, and processes.

Plug-ins

Plug-ins provide basic processing functions as well as integration with third-party tools. Serena Release Automation ships with plug-ins for several common deployment processes, and others are readily available for a wide variety of tools, such as middleware tools, databases, servers, and other deployment targets.

Third-party tools exhibit wide and varied functions, of course. Plug-in integration is achieved by breaking down a tool's functions into simple, discrete steps that invoke a specific behavior. A plug-in step might invoke a tool, or invoke different functions in a tool, such as extracting or inserting some type of data.

When you use plug-ins to create a component process, you can use steps from several plug-ins and configure the steps as you go. For example, you might create a process using a plug-in for a source control tool that deploys a component to a middleware server, and another plug-in to configure a step that removes the component from the server.

A component process that contains a plug-in step requires an agent. Unless the agent needs to interact with the host's file system or system processes, the agent does not have to be on the same host as the target resource.

Serena Release Automation enables you to download and install numerous component plug-ins. Serena does not charge any additional fees for plug-ins. The plug-in system is open and extensible—plug-ins can be written in any language.

Component Versions and the CodeStation Repository

After defining a component's source and processes, you import its artifacts into Serena Release Automation'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). This provides several benefits, such as tamper-proof storage, and the ability to review and validate artifacts with Serena Release Automation's user interface. But if you have storage concerns or use a tool like Maven, you can limit CodeStation to using references to the artifacts instead of actually copying them.

Each time a component is imported, including the first time, it is versioned. Versions can be assigned automatically by Serena Release Automation, 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.

Applications

An application is the mechanism that initiates component deployments; they bring together components with their deployment targets, and orchestrate multi-component deployments.

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 controlled environments as well. As the application moves components through various environments, Serena Release Automation 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.

Agents

An agent is a process that runs on target host and communicates with the Serena Release Automation server. Agents are integral to Serena Release Automation's client/server architecture. Agents perform the actual work of deploying components and so relieve the server from the task, making large deployments involving thousands of targets possible.

Typically, an agent runs on the same host where the resources it handles are located. A single agent can handle all resources on its host. If a host has several resources, an agent process is invoked separately for each resource. For example, a test environment might contain a single web server, a single middleware server, and a single database server all running on the same host (machine). A deployment to this environment might have one agent and three separate resources.

Depending on the number of hosts in an environment, a deployment might require a large number of agents. Agents are unobtrusive and secure. Agent communications use SSL encryption and mutual key-based authentication. For added security, agents do not listen to ports, but open direct connections to the server instead.

Resources

A resource is a user-defined construct based on Serena Release Automation's architectural model. Resources aid bookkeeping; inventory is tracked for resources. Resources are created and managed through the user interface.

A resource represents a deployment target—a physical machine, virtual machine, database, J2EE container, and so on. Components are deployed to resources by agents (which are physical processes). Resources generally reside on the same host where its managing agent runs. A host can have more than one resource. If an agent is configured to handle multiple resources, a separate agent process is invoked for each one.

A resource can represent a physical machine, which is the simplest configuration, or a specific target on a machine, such as a database or server. So a host (machine) can have several resources represented on it. In addition, a resource can represent a process distributed over several physical or virtual machines. Finally, environments consist of resources.

To perform a deployment, at least one resource must be defined and (usually) at least one agent. ("Usually" because trivial deployments can be done without an agent.) Typically, each host in a participating environment has an agent running on it to handle the resources located there.

A proxy resource is a resource effected by an agent on a host other than the one where the resource is located. If an agent does not require direct interaction with the file system or with process management on the host, a proxy resource can be used. When a deployment needs to interact with a service exposed on the network (a database or J2EE server, for instance), the interaction can happen from any machine that has access to the networked service.

Resource Groups

A resource group is a logical collection of resources. Resource groups enable collections of resources to be easily reused. Resource groups can manage multi-tenant scenarios, for example, in which several machines share the same resources.

Architecture

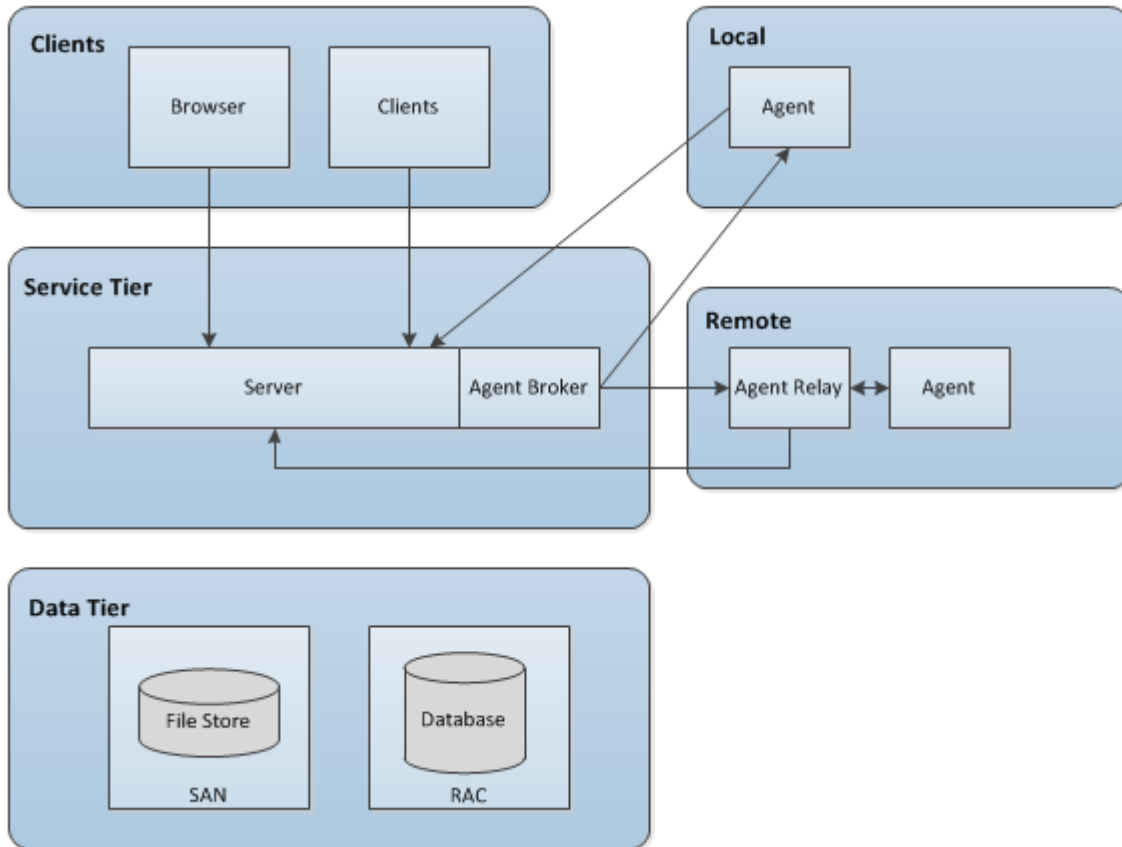
Serena Release Automation architecture consists of a service tier and a data tier. The service tier has a central server that provides a web server front-end and core services, such as workflow, agent management, deployment, inventory, security, as well as others. A service can be thought of as a self-contained mechanism for hosting a piece of business logic. Services can be consumed by clients\agents or other services. Deployments are orchestrated by the server and performed by agents distributed throughout the network. Most clients use browsers to communicate with the web server via HTTP(S). Most server-agent communication is done via JMS (discussed below) but HTTP(S) is also used as required.

Serena Release Automation uses stateless communications for server-agent communications (JMS-based) as well as client-web service communications. Stateless, as used here, means the server retains little session information between requests and each request contains all the information required to handle it. The server sets-up listening sockets and listens for agents, relays, and users (clients). For added security, agents do not listen on ports. Agents send requests when they are ready to make the transition to a new state.

Server-agent communication is built around transferring—deploying—components. Components can contain any business-meaningful content, such as environment information, configuration data, source, static files, or anything else associated with a software project. Because JMS connections are persistent and not based on a request-response protocol, Serena Release Automation does not have to continually open and close ports, which enables the server to communicate with agents at any time while remaining secure and scalable.

Many Serena Release Automation services are REST-type (representational state transfer). REST-style services are web services that focus on transferring resources over HTTP. A resource can be any business-meaningful piece of data. Resources are transferred by a self-describing format such as XML or JSON (JavaScript Object Notation). The XML and JSON representations typically model resource states at the time of agent/client requests. REST-style services achieve statelessness by ensuring that requests include all the data needed by the server to make a coherent response.

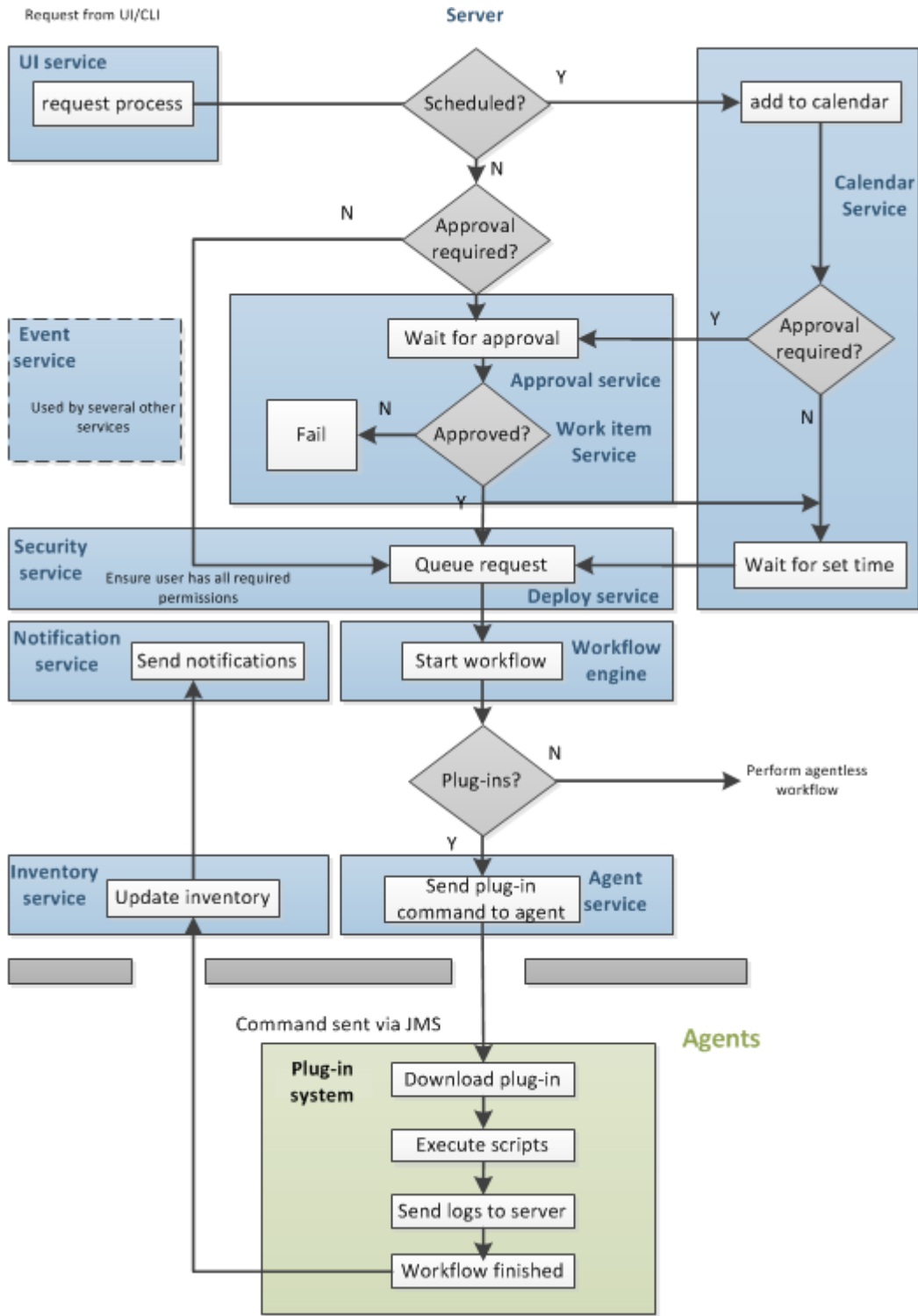
The data tier's relational database stores configuration and run-time data. The data tier's file store—CodeStation—contains log files, artifacts, and other non-structured data objects. Reporting tools can connect directly to the relational database.



Service Tier

The Serena Release Automation server provides a variety of services, such as: the user interface, component and application configuration tools, workflow engine, and security services among others. The REST-based user interface provides the web-based front-end that is used to create components and fashion workflows; request processes, and manage security and resources, among other things.

When a workflow is requested, many services are used to fulfill the request, which are shown in the following illustration:



Workflow requests are initiated with the user interface, either the web-based application or the CLI (command line interface).

Services Table

Service	Description
User Interface	Used to create components and fashion workflows, request processes and manage security and resources, among other things. REST-based.

Service	Description
Workflow Engine	Manages workflows—application and component processes. Calls the agent responsible for performing the workflow's current plug-in step. When the workflow is finished, alerts the notification and inventory services. Called by the deploy service. REST-based.
Agent	Tracks installed agents and routes plug-in commands to affected agents. Commands come from plug-in steps. Invoked by the workflow service. REST-based.
Work Item	Operates in tandem with the approval service; provides approver alerts and enables approvers to accept or reject workflows. If a scheduled workflow remains unapproved at run-time, the job fails automatically. REST-based.
Plug-in Manager	Serena Release Automation can interact with virtually any system through its extensible plug-in system; plug-ins provide functions by breaking-down tool features into automated steps. Plug-ins can be configured at design- and run-time. When a plug-in step executes, the controlling agent invokes its run-time process to execute the step. When a new component version is available, the agent compares the current component version and downloads and only new or changed artifacts.
Event	The event service is ubiquitous; it alerts other services as various trigger conditions occur.
Deployment Service	Manages deployments. When a deployment process is requested, invokes the workflow engine to perform the process. Works in tandem with the security service to ensure users have required permissions. REST-based.
Notification Manager	Notifies users about the status of deployments; notifications are sent to approvers if the system is configured with an email server and the user has an email address. Invoked by the workflow manager. REST-based.
Inventory Manager	When a workflow finishes, the inventory manager updates affected inventory records. Serena Release Automation maintains an inventory of every deployed artifact in every environment, which provides complete visibility across environments. REST-type service.
Approval Engine	Enables creation of approval-requiring jobs and approver roles. Works in tandem with the work item service to ensure required approvals are made before scheduled jobs. REST-based.
Security	Controls what users can do and see; maps to organizational structures by teams, roles, activities, etc. REST-based.
Calendar	Used to schedule processes to being at some future point; works in tandem with the approval and work item services. REST-based.
CodeStation	Handles versioning of artifacts; agents invoke it when downloading component versions. REST-based.

Clients

Web browsers are Serena Release Automation's most common client (agents are discussed in another topic, see the [Agents \[page 93\]](#) section), but other clients can be developed to access the web services. Clients are deployed locally (on the same LAN as the Serena Release Automation server) or remotely, and communicate with the server via HTTP or HTTPS. The Serena Release Automation browser-based GUI is a Rich Internet Application (RIA) that maintains much of its functionality in the browser. Clients interact with RESTful (representational state transfer) services on the server as needed. A command line client is available that provides most of features found in the browser-based GUI. The command line client is also built on top of RESTful services.

Relational Database

Your relational database is a critical element for performance and disaster recovery. The provided Derby database, while sufficient for proof-of-concept work, is generally insufficient for the enterprise. Full-featured databases like Oracle, MS SQL Server, or MySQL are better options. Ideally, the database—whichever is used—should be configured for high-availability, high-performance, and be backed-up regularly.

10-20 GB of database storage should be sufficient for most environments. For Oracle, an architecture based on Oracle RAC is recommended; for Microsoft SQL Server, a clustered configuration is preferred; for MySQL, utilize MySQL Cluster.

File Storage—Codestation

The data tier also provides log file and Codestation artifact storage. Artifacts represent deployable items such as files, images, databases, configuration materials, or anything else associated with a software project. By default, these are stored in the `var` subdirectory in the Serena Release Automation server installation directory. In an enterprise environment, the default installation might not be ideal, see [Relocating Codestation](#) for a discussion about enterprise options.

Serena Release Automation's secure and tamper-proof artifact repository ensures that deployed components are identical to those tested in preproduction environments. Without the repository, artifacts would have to be pulled from network shares or some other system, increasing both security risks and the potential for error.

The artifact repository uses content addressable storage to maximize efficiency while minimizing disk use. The repository tracks file versions and maintains a complete history for all components. Maximizing efficiency is important, since the artifact repository stores files that are much larger than source files. Association of files with Components is built into the system. Without any configuration, each Component gets its own area of the repository for its files. There is no chance of confusion or mix-up of files to Components. And, each Component Package is mapped to a specific set of files and versions corresponding to the Component.

The artifact repository comes with a client application that provides remote access to the repository. Using the client, the user can add/modify files, create Packages, retrieve files, as well as view the history of changes. The client application provides a file transfer capability that can be used to deliver files to target servers during deployments. This built-in transfer mechanism verifies the integrity of all transferred files against their expected cryptographic signatures, thus guaranteeing that files have not been corrupted during transmission or tampered with during storage. In addition to the client application, the artifact repository exposes REST-based web services. These services are used to build integrations between build systems such as AnthillPro and Serena Release Automation. Such integrations automatically place the artifacts produced by the build process in the artifact repository, thus making the artifacts available for deployment.

Data Center Configuration

Installation recommendations.

Cold Standby

Serena Release Automation employs the cold standby HA strategy for the application tier. When the primary system fails, the cold standby is brought online and promoted to primary server. Once online, the standby reestablishes connections with all agents, performs recovery, and proceeds with any queued processes. Because the most intense work is handed-off to agents, a high performance configuration should not have an agent installed on the same hardware as the main server.

The Serena Release Automation server aggressively utilizes threading and takes advantage of any additional CPU cores assigned to it. A small to midrange server with 2-4 multi-core CPUs is ideal, but, because it is relatively easy to move an existing Serena Release Automation server installation to a new machine, starting small and scaling as needed is a very legitimate strategy. The memory available to the application tier should also be increased from the default 256 MB to something on the order of 1 GB.

Platform Considerations

Serena Release Automation agents are platform agnostic, and can be installed on anything that provides a Java 1.5 JDK. The server process is also platform agnostic. Our customer base includes large Serena Release Automation installations on Windows, Solaris, AIX, HP-UX, other Unix flavors and various Linux platforms, all running successfully.

We have seen somewhat better performance from Unix and Linux operating systems, but recommend installing on the platform with which you are most familiar and comfortable.

Recommended Server Installation

- **Two server-class machines**

Serena 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**

2-4 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 Serena Release Automation's artifact repository (CodeStation), the more storage needed.

- **Network**

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

Agent Minimum Requirements

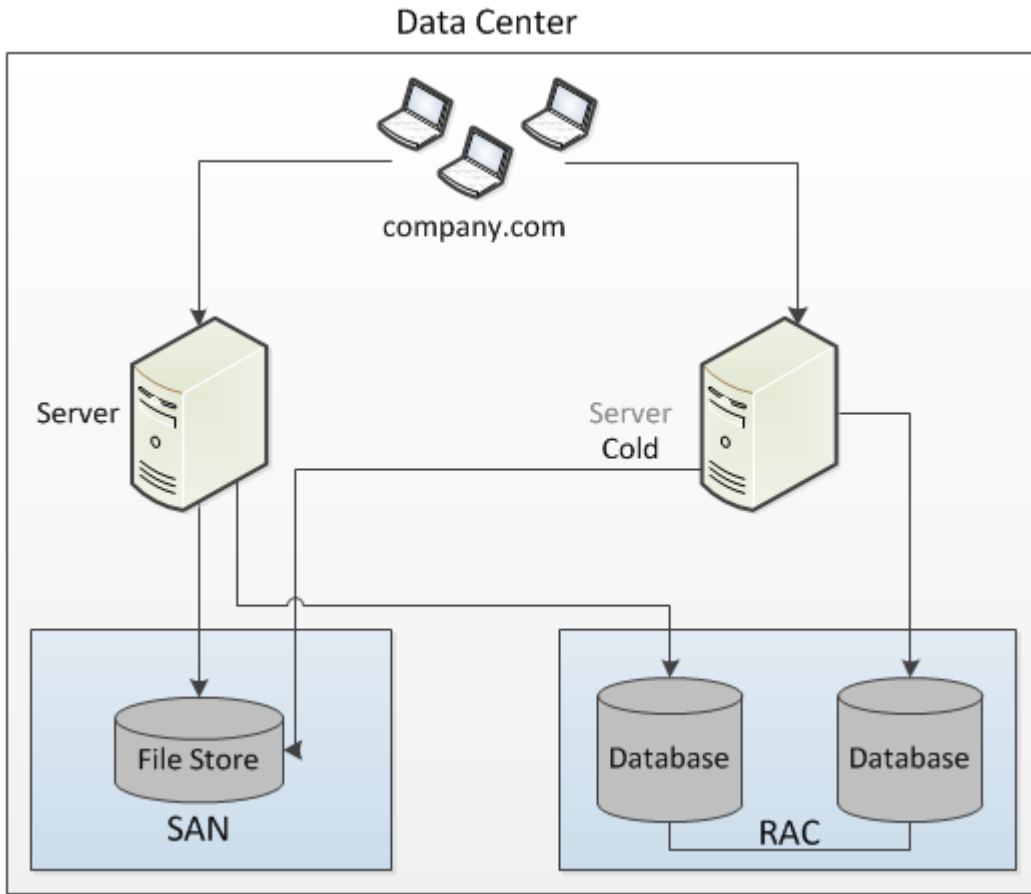
Designed to be minimally intrusive, agents require 64-256 MB of memory and 100 MB of disk space. Additional requirements are determined by the processes the agent will run. Agents should be installed on separate machines. For evaluation purposes, a good option is to install an agent on a virtual machine.

Typical Data Center Configurations

Most organizations configure the data tier with network storage and a clustered database. The service tier performs best when it's on a dedicated, stable, multi-core machine with a fast connection to the data tier. A standby machine should be maintained and kept ready in case the primary server goes down.

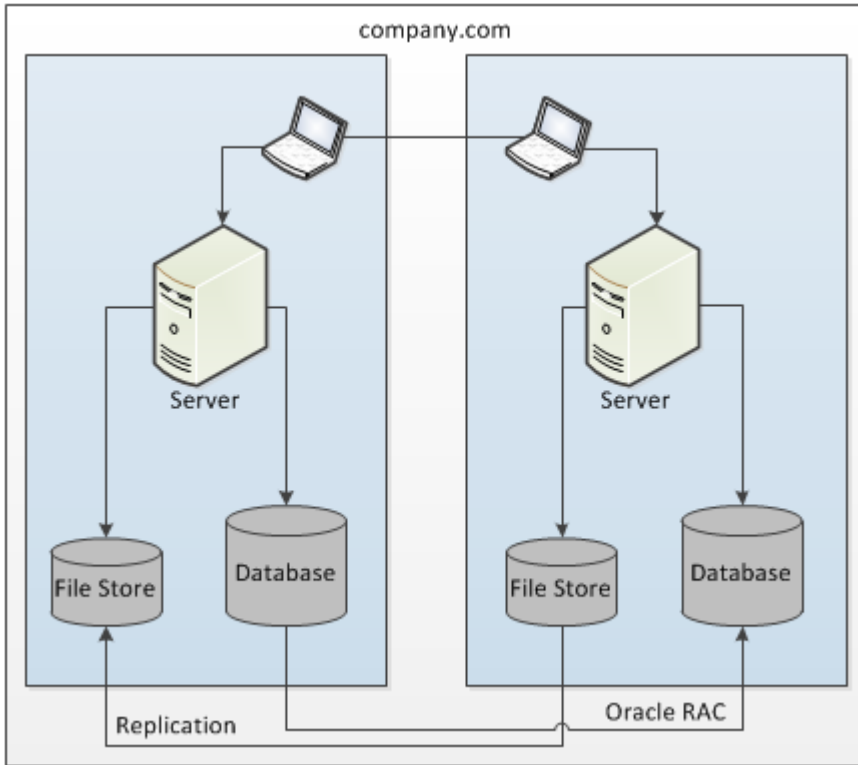
The following figures illustrate typical Serena Release Automation configurations.

Single Data Center Configuration



There are no remote agents or agent relays in this configuration.

Multiple Data Centers

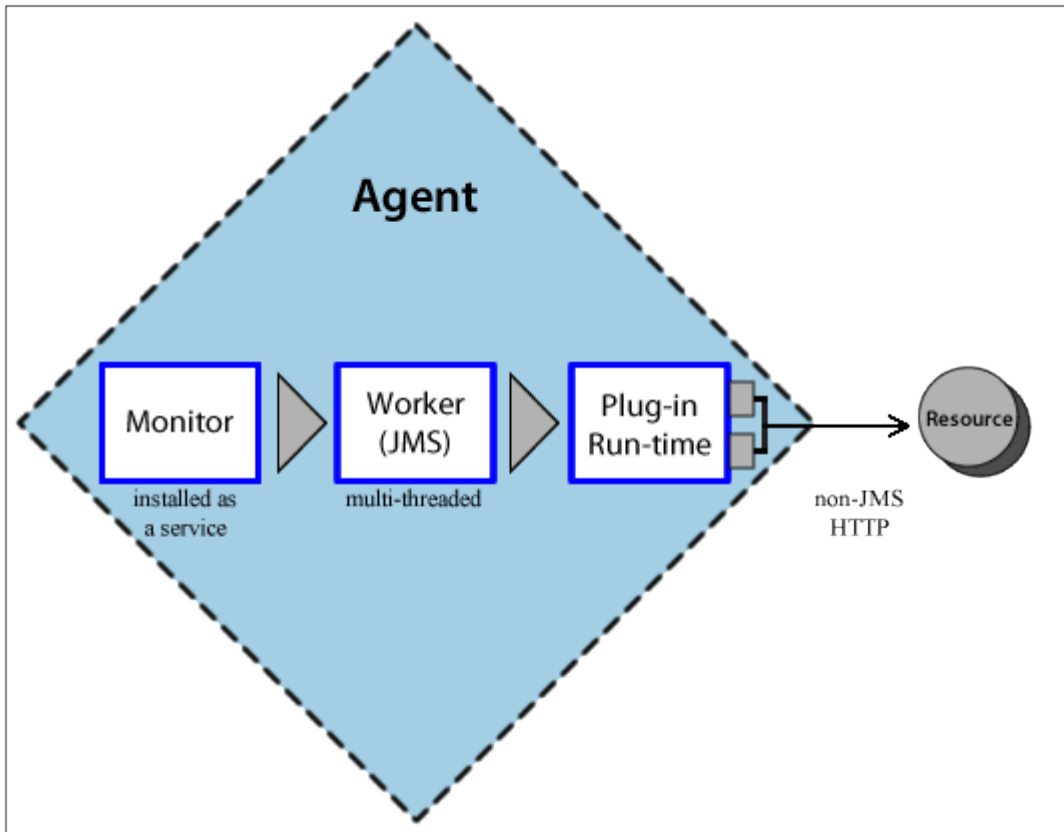


Agents

Agents play a central role in the Serena Release Automation architecture. An agent is a lightweight process that runs on a deployment-target host and communicates with the Serena Release Automation server. Agents perform the actual work of deployment which relieves the server from the task. All processes—packaging, configuration, deployments, and so on—requested by the Serena Release Automation server are executed on hardware assigned to agents. Once an installed agent has been started, the agent opens a socket connection to the Serena Release Automation server. Communication between server and agents uses a JMS-based (Java Message Service) protocol and can be secured using SSL, with optional mutual key-based authentication for each end-point. This communication protocol is stateless and resilient to network outages (the benefits of statelessness are discussed below).

While we characterize an agent as a single process, technically an agent consists of a *worker* process and a *monitor* process. The worker is a multi-threaded process that performs the actual deployment work after receiving commands from the server. Work commands come from plug-in steps which provide seamless integration with many third-party tools. The monitor is a service that manages the worker process—starting and stopping, handling restarts, upgrades, and security, for example. Agents are rarely upgraded because their functionality is derived from plug-ins, which can be upgraded at will. Once an agent is installed, it can be managed from the Serena Release Automation web application.

Agent Processes



Agents are an important part of Serena Release Automation's scalability. By adding more agents, the throughput and capacity of the system increases almost exponentially and so can scale to fit even the largest enterprise.

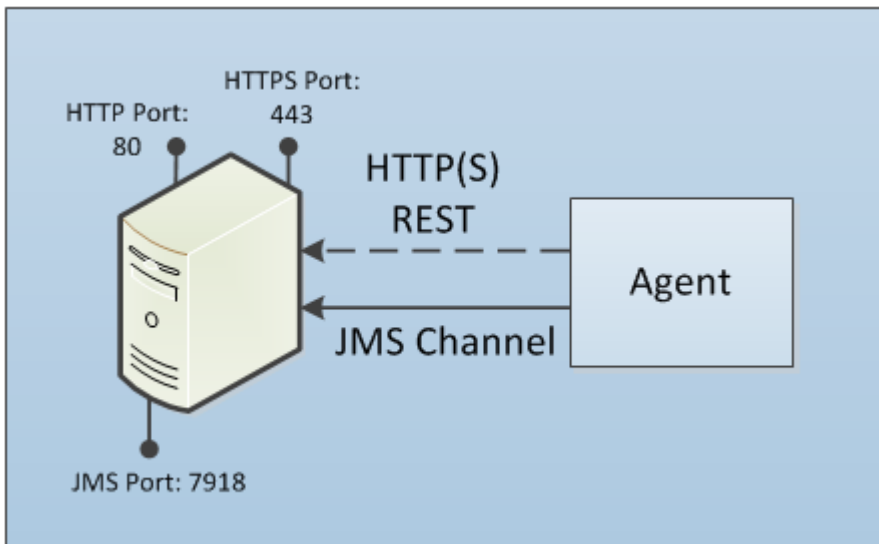
Server-Agent Communication

Most agent communication is done with JMS, but some agent activities—posting logs, transmitting test results, or posting files to Codestation, for example—use the web tier via HTTP(s) as needed. The JMS channel is Serena Release Automation's primary control channel; it's the channel the server uses to send agent commands. By default the server listens on only three ports: port 7918 for JMS, 8080 for HTTP, 8443 for HTTPS.

The agent monitor service uses JMS for all server communications and for sending commands, such as "run step," to the worker process. The worker process uses JMS for system communications, and HTTP REST services when performing plug-in steps or retrieving information from the server.

Stateless server-agent communication provides significant benefits to performance, security, availability, and disaster recovery. Because each agent request is self-contained, a transaction consists of independent message which can be synchronized to secondary storage as it occurs. Either endpoint—server or agent—can be taken down and brought back up without repercussion (other than lost time). If communications fail mid-transaction, no messages are lost. Once reconnected, the server and agent automatically determine which messages got through and what work was successfully completed. After an outage, the system synchronizes the endpoints and recovers affected processes. The results of any work performed by an agent during the outage are communicated to the server.

Stateless Communication



In the "Stateless Communication" [page 25] figure, the arrow represents the direction in which communication was established, but the flow can be in both directions with JMS.

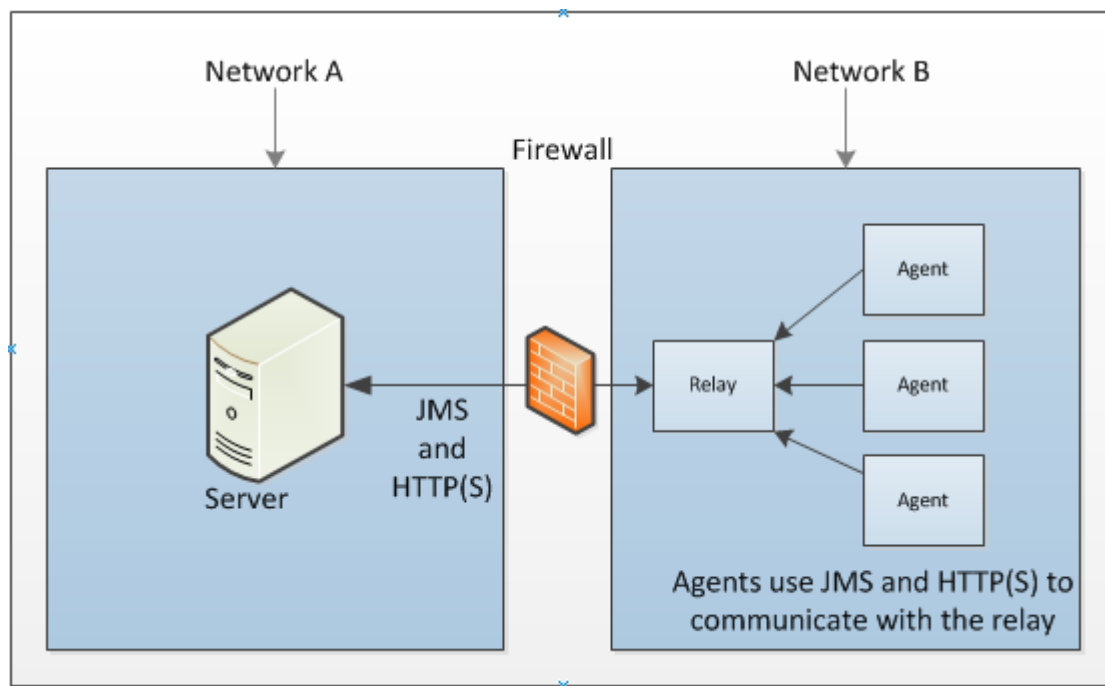
Remote Agents: Crossing Network Boundaries and Firewalls

Serena Release Automation supports remote agents cross-network deployments. As long as there is at least a low bandwidth WAN connection between the server and remote agents, the Serena Release Automation server can send work to agents located in other geographic locations. To aid performance and ease maintenance, Serena Release Automation uses agent relays to communicate with remote agents. 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. Agent relays can be configured as CodeStation proxies in order to optimize the transfer of large objects.

The following, a simple artifact move, illustrates the mechanics of remote communications:

1. A remote agent starts and establishes a connection to the agent relay via JMS, which, in turn, alerts the Serena Release Automation server via JMS that the remote agent is online.
2. The server sends, say, an artifact download command to the relay via JMS, and the relay delivers the message to the remote agent (also via JMS).
3. The server locates the artifacts, and then:
 1. Uploads the artifacts to the relay over HTTP(s), which begins streaming them directly to the agent over the server-relay HTTP(s) connection.
 2. Once the remote agent completes the work, it informs the server via JMS.

Crossing Network Boundaries



By default, agent relays open the connection to the Serena Release Automation server, but the direction can be reversed if your firewall requires it. Remote agents open connections to the agent relay.

In configurations with relay agents, agents local to the Serena Release Automation server continue to use direct communications.

Agent Security

Serena Release Automation agents employ user impersonation when required to perform tasks for which they would not otherwise have permission. To run a database update script, for example, an agent might need to be the "oracle" user; but to update the application, the agent might need to be the "websphere" user. By using impersonation, the same agent can run the script and update the application, which enables you to combine these steps into a single process. For information about user impersonation, see the ["User Impersonation" \[page 26\]](#) section.

User Impersonation

Serena Release Automation can use user impersonation when an agent must execute a command for which it might not otherwise have permission, or when a specific user must be employed for a given process. On Unix/Linux systems the `su/sudo` commands are used to impersonate users; on Windows Serena Release Automation provides a utility program to handle impersonation. You implement impersonation when you configure a component's plug-in process step.

Using `su/sudo`

The `su` command (as used by Serena Release Automation) enables a user to start a shell as another user (process steps can be considered individual shells). When you configure a process step (see [Process Editor \[page 75\]](#)), you can tell Serena Release Automation to use impersonation for the step. By default, `su` is used but you can use `sudo` instead. To configure impersonation, you supply the user name required by the target host. When the impersonation-configured process step runs, the `su` or `sudo` command runs the step as the impersonated user. Each step that needs user impersonation must be configured independently.

Before `sudo` can be used, impersonation privileges must be defined in the `/etc/sudoers` file. When you configure `sudoers`, ensure that the impersonating user does not have to supply a password. Typically, you would configure the `/etc/sudoers` file like this:

```
Defaults:X !requiretty
```

```
X ALL=(Y) NOPASSWD: ALL
```

where X and Y are user names. Configured this way, user X can run any command as user Y without supplying a password.

`su` and `sudo` maintain a record in the system logs of all of their activity. `su` can be used without configuring the `sudoers` file. For information about `su/sudo` see the Unix/Linux documentation.

NOTE

For UNIX- or Linux-based agents the password option is ignored.

Impersonation on Windows Systems

For agents running on Windows platforms, Serena Release Automation provides a program that handles impersonation. You implement impersonation for Windows-based agents the same way you do for UNIX- or Linux-based agents: when you configure a process step, you specify the *local user* credentials—user name and password—that will be used when the step is processed. For impersonation purposes, a local user is one whose user name and password are stored on the target computer and who is part of the administration group and has, at a minimum, 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` (local logon)

You can also impersonate the Windows LocalSystem account. The LocalSystem account is installed on every Windows machine and is the equivalent of the root user on Unix/Linux. It is guaranteed to have the privileges listed above.

NOTE

For Windows-based agents the `sudo` option is ignored if selected.

SSL Mutual Key-based Authentication

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.

SSL technology can be used in several modes. In *unauthenticated mode*, communication is encrypted/decrypted but users do not have to authenticate or verify their credentials. By default Serena Release Automation uses this mode for its JMS-based server/agent communication. By default, JMS-based communication uses port 7918.

SSL unauthenticated mode can also be used for HTTP communication. You can implement this mode for HTTP during server/agent installation, or activate it afterward. See [SSL Configuration \[page 46\]](#).

In *mutual authentication mode*, communications are encrypted as usual, but users are also required to authenticate themselves by providing digital certificates. A digital *certificate* is a cryptographically signed document intended to assure others as to the identity of the certificate's owner. Serena Release Automation certificates are self-signed.

When mutual authentication mode is active, Serena Release Automation uses it for JMS-based server/agent communication. In this mode, the Serena Release Automation server provides a digital certificate to each agent, and each agent provides one to the server. This mode can be implemented during server/agent installation, or

activated afterward. See [SSL Configuration \[page 46\]](#) for information about activating this mode and exchanging certificates between the server and agents.

Unauthenticated mode for HTTP and mutual authentication mode for JMS are optional; you can implement one without implementing the other, or implement both.

Getting Started

This documentation contains the following important information you will need to get you started with Serena Release Automation:

- [Serena Release Automation Roadmap \[page 29\]](#)
- [Installing Servers and Agents \[page 32\]](#)
- [Quick Start–helloWorld Deployment \[page 50\]](#)

Serena Release Automation Roadmap

This section provides information that will help you quickly become familiar with Serena Release Automation, starting with installation and configuration. Then, descriptions of how to create components and define applications to deploy them, and finally how to perform deployments. The following topics should be reviewed in order:

- Creating components, see [Creating a Component \[page 30\]](#)
- Creating applications, see [Creating an Application \[page 30\]](#)
- Deploying components, see [Deploying a Component \[page 31\]](#)

Installing Serena Release Automation

A basic configuration consists of a server, a database, and at least one agent. In production environments, all three should be installed on separate machines.

The following table summarizes basic installation steps. Related topics are listed below the table.

Installation Steps table

Step	Description
1. Review installation recommendations	Requirements and recommendations, including performance recommendations, are provided.
2. Download Serena Release Automation installation files	Download the server, agent, agent relay, and CLI client (command line interface) installation packages. Installation files can be downloaded from the Serena Release Automation support portal http://support.serena.com/Case/CaseHome.aspx . 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. Create an empty database for Serena Release Automation. Serena Release Automation supports Oracle, MySQL, and Microsoft SQL Server. For installation information, see Database Installation [page 35] .
3. Install the database	NOTE The installation package includes a lightweight database—Derby—that can be used for evaluation purposes. For installation information, see Server Installation [page 37] . 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 <code>installed.properties</code> file located in the <code>server_install/conf/server/</code> directory. If you intend to turn on SSL, see SSL Configuration [page 46] .
4. Install the server	

- Agents are installed on target machines and communicate with the server. When installing an agent, you supply several values defined during server installation. See [Agent Installation \[page 43\]](#) for instructions about installing agents. An agent requires various access privileges for the machine where it is installed, which are described in that section.
5. Install agents
6. Confirm installation
- Start the server and agents. For information about running the product, see [Running Serena Release Automation \[page 49\]](#). 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.

Creating a Component

Components are the centerpiece of Serena Release Automation'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.

Component Creation Steps table

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 Creating Components [page 67] .
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). Serena Release Automation 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 Serena Release Automation provides plug-ins for many common functions. Numerous other plug-ins are available from Serena— http://support.serena.com .

Related topics:

- How to create manual tasks, see [Component Manual Tasks \[page 85\]](#)
- How to install plug-ins, see [Installing Plug-ins \[page 146\]](#)
- How to create and use templates, see [Creating a Component Template \[page 86\]](#)
- How to import component templates, see [Importing Templates \[page 87\]](#)

Creating 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.

Application Creation Steps table

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:

- [Serena Release Automation properties \[page 164\]](#)
- [How to create snapshots \[page 15\]](#)
- [How to import applications \[page 98\]](#)

Deploy the Component

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

Deployment Steps table

Step	Description
1. Select environment [page 110]	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.
2. Run process [page 110] s	You run a process by selecting it for a given environment and specifying certain other parameters. Processes can also be run with the CLI [page 166] , or scheduled for a future time [page 113] .
3. Check results [page 112]	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 \[page 144\]](#)
- [How to setup authorizations \[page 47\]](#)
- [How to create application gates \[page 107\]](#)

Other Topics

The following list provides links to additional topics.

- [How to setup security \[page 128\]](#)
- [How to run reports \[page 114\]](#)
- [How to use the command line interface \[page 166\]](#)
- [How to create plug-ins \[page 155\]](#)
- [How to add agents to a product license \[page 140\]](#)

Installing Servers and Agents

A Serena Release Automation installation consists of the Serena Release Automation 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 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 Serena Release Automation in a production environment, Serena recommends the use one of the supported databases (Oracle Database (all versions), SQL Server, or MySQL).

Installation Steps

1. Review the system requirements. See [System Requirements \[page 33\]](#).
2. Ensure that Java is installed on the server machines (and agent relay machine if used). All server and agent relay 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.

NOTE

This step does not apply to agent machines because the agent installer installs a JRE.

3. Download the Serena Release Automation server and agent installation files from the Serena support portal (see [Downloading Serena Release Automation \[page 34\]](#)).
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 [Database Installation \[page 35\]](#).
6. Complete database installation by configuring the appropriate JDBC driver (typically supplied by the database vendor).
7. Create an empty database for Serena Release Automation and at least one dedicated user account.
8. Install the server. See [Server Installation \[page 37\]](#).
9. If you are using an agent relay, install the relay. See [Installing Agent Relays \[page 44\]](#).
10. Finally, install at least one agent. See [Agent Installation \[page 43\]](#).

For information about using the command line interface (CLI), see [Command Line Client \(CLI\) Reference \[page 166\]](#).

For information about running the installed items and accessing the Serena Release Automation web application, see [Running Serena Release Automation \[page 49\]](#).

Installation Recommendations

Important

Except for evaluation purposes, do *not* install an agent on the same machine as the server.

Many Serena Release Automation users have found that by following these general guidelines you can 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, Serena Release Automation 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 Serena Release Automation. Because Serena Release Automation 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 affect the host file system.
- **Except for evaluation purposes, do not install an agent on the Serena Release Automation server machine.** Because the agent is resource intensive, installing one on the server machine can degrade performance during large deployments.
- **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

Serena Release Automation 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 Serena Release Automation.
- Java version: JRE 5 or greater.

Server Installation Recommendations

- **Two server-class machines**

Serena 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 Serena Release Automation's artifact repository (CodeStation), the more storage needed.

NOTE

CodeStation is installed when the Serena Release Automation 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 Serena Customer 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.

IMPORTANT!

In production environments, Serena *highly recommends* installing agents on separate machines.

32- and 64-bit JVM Support

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

JVM Support table

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

Downloading Serena Release Automation

The installation package is available from the Serena support portal. If you need help accessing the portal, contact your Serena Sales representative.

NOTE

You must have a license in order to download the product. For an evaluation license, please contact your Serena Sales Representative.

1. Go to <http://support.serena.com/Case/CaseHome.aspx> and log in using your customer account.

If you do not have an account, please contact your Sales Representative.

2. Browse to the My Downloads tab.
3. From the **Please Select Product** drop-down, select **Serena Release Automation**.
4. Select the Serena Release Automation version you want to download.
5. Select the appropriate package for your environment for the server, agent, and agent relay.

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

A license file should have been provided to you by your Serena Sales Representative.

Database Installation

Currently, Serena Release Automation supports Derby, Oracle, SQL Server, and MySQL.

Installing Oracle

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

When you install Serena Release Automation, you will need:

- the Oracle connection information, and
- a user account with table creation privileges.

Serena Release Automation 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 an Oracle database:

1. Download and install the JDK—not a JRE—that corresponds to your operating system (see <http://www.oracle.com/technetwork/java/javase/downloads/>).
2. Download the Oracle JDBC driver specific to the database edition you are using (see <http://www.oracle.com/technetwork/indexes/downloads/index.html>) and save in your Apache-Tomcat `libs` folder.
3. Install the JDBC driver.
4. Create the Oracle database by executing the following commands:

```
CREATE USER serena_ra;  
  
GRANT CONNECT, RESOURCE serena_ra
```

5. Begin server installation, see [Server Installation \[page 37\]](#).

Select the Oracle database option in the installer.

6. Provide the Oracle JDBC driver (see [step 2 \[page 35\]](#)).
7. Provide the JDBC driver class Serena Release Automation will use to connect to the database.

The default value is:

```
oracle.jdbc.driver.OracleDriver
```

8. 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
```

9. Finish by entering the database username and password.

NOTE

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

Installing MySQL

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

When you install Serena Release Automation, you will need the MySQL connection information, and a user account with table creation privileges.

To install the MySQL database:

1. Create a database by executing the following commands:

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

2. Obtain the MySQL JDBC driver.

The JDBC jar file is included among the MySQL installation files. The driver is unique to the edition you are using.

3. Begin server installation, see [Server Installation \[page 37\]](#).

Select the MySQL database option in the installer.

4. Provide the MySQL JDBC driver (see [step 2 \[page 36\]](#)).

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

5. Provide the JDBC connection string. Typically, it is similar to:

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

For example:

```
jdbc:mysql://localhost:3306/Serena Release Automation
```

6. Finish by entering the database username and password.

Installing Microsoft SQL Server

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

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

To install the MS SQL Server database:

1. Download and install the JDK—not a JRE—that corresponds to your operating system (see <http://www.oracle.com/technetwork/java/javase/downloads/>).
2. Download the SQL Server JDBC driver from <http://www.microsoft.com/en-gb/download/> and save it in your Apache-Tomcat `libs` folder.
3. Install the JDBC driver.
4. Using the SQL Server Management Studio, execute the following commands:

```
CREATE DATABASE serena_ra;

USE serena_ra;

CREATE LOGIN serena_ra WITH PASSWORD = 'password';

CREATE USER serena_ra FOR LOGIN serena_ra WITH DEFAULT_SCHEMA = serena_ra;

CREATE SCHEMA serena_ra AUTHORIZATION serena_ra;

GRANT ALL TO serena_ra;
```

5. Begin server installation, see [Server Installation \[page 37\]](#).
6. Select the SQL Server database option in the installer.
7. Provide your SQL Server database details:

MS SQL Server Database Installation Details table

Field	Description
JDBC driver jar filename	Browse to and select the JDBC driver jar file you downloaded in step 2 [page 37] .
JDBC driver class	The default value is: <code>com.microsoft.sqlserver.jdbc.SQLServerDriver</code>
Database connection string	The format depends on the JDBC driver. Typically, it is similar to: <code>jdbc:sqlserver://[DB_URL]:[DB_PORT];databaseName=[DB_NAME]</code> For example, <code>jdbc:sqlserver://localhost:1433;databaseName=Serena Release Automation</code>
User	Database username. The default is <code>serena_ra</code> .
Password and Confirm password	Database username password, and re-entry for confirmation.

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.

IMPORTANT!

If you are installing the server in a production environment, install and configure the database you intend to use *before* installing the server. See [Database Installation \[page 35\]](#).

Interactive Server Installation (Windows, Linux/UNIX (AIX, Solaris))

NOTE

(Linux 64bit systems only) The ia32-libs package must be installed for 32bit compatability mode; the Serena Release Automation server installer is a 32bit application.

This type of installation utilizes a wizard that guides you through the complete process. The properties set during the server installation are recorded in the `installed.properties` file located in the `server_install/conf/server/` directory.

To install the server:

1. Download the installation files. For more information, see [Downloading Serena Release Automation \[page 34\]](#).
2. Run the downloaded installer `SerenaRA-Server.exe`, or your platform-specific alternative, and follow the step-by-step instructions.

Server Installation (Other UNIX Platforms)

These server installation instructions are for UNIX platforms other than Linux/UNIX (AIX, Solaris).

To install the server:

1. Download the `serena_ra.war` file. This file is contained in the `Serena.RA-Other40.zip`. For more information, see [Downloading Serena Release Automation \[page 34\]](#).
2. Stop Apache Tomcat.
3. Copy and paste the `serena_ra.war` file into your Tomcat 1.6 or Serena Common Tomcat `webapps` folder.
4. Restart Tomcat.
5. Launch a browser and navigate to http://localhost:8080/serena_ra.

The configuration setup displays.

6. Modify any configuration parameters as needed:
 - Install location
 - External Web URL
 - Agent port and choice of mutual authentication
 - Database (port, schema, user, password)
7. Click Install.

When the server installation is complete, the Serena Release Automation login page displays.

Silent Mode Server Installation

This section contains information about how to implement a silent install of the Serena Release Automation server on Windows, Linux/UNIX (AIX, Solaris).

(Windows) Server Silent Installation

To install the server in silent mode:

1. Download the installation files. For more information, see [Downloading Serena Release Automation \[page 34\]](#).
2. Create an options file and save as `C:/optionsFile.txt` (see [\(Windows\) optionsFile.txt \[page 39\]](#)).
3. In Windows, issue the command:

```
SerenaRA-Server.exe /s /V"/qn /L*vx "%TEMP%\silent-install.log"
PROFFILE="c:\optionsFile.txt" "
```

Where:

- monospace; ">"%TEMP%\silent-install.log" is an absolute path to a logfile.
- PROFFILE="c:\optionsFile.txt\" is an absolute path to the options file you created in [Step 2 \[page 39\]](#).
- You have included the **required** space between the final two quotes.

For examples of the `optionsFile.txt`, see [\(Windows\) DB Examples - Server Silent Install \[page 39\]](#).

(Windows) Server Silent Install Options

Option	Default	Description
AgreeToLicense	Yes	must be set to Yes
SRA_USER_INSTALLDIR	c:\Documents and Settings\Administrator\.serena\ra	directory to install the Serena Release Automation server.
DB_TYPE	derby	Use to specify a database vendor other than the default: MYSQL ORA SQLSVR
<DB>_USER	password	database user ID, where <DB> is: DERBY MYSQL ORA SQLSVR
<DB>_PASSWORD	password	password for database user ID, where <DB> is: DERBY MYSQL ORA SQLSVR
ORA_DB_SCHEMA	serena_ra	(Required for Oracle) database schema
<DB>_JDBC_DRIVER		(Required)For database other than Derby, JDBC database driver file, where <DB> is DERBY MYSQL ORA SQLSVR
DERBY_PORT	11377	(Required for Derby) specify the Derby port.
<DB>_DB_CONN		(Required)For database other than Derby, database connection, where <DB> is MYSQL ORA SQLSVR
AGENT_MUTUAL_AUTH	N	agent mutual authentication option: Y N
JMS_PORT	7918	server port
IS_INSTALL_MODE	silent	must be set to: "silent"
TC_PORT	8080	Tomcat port. Only used for new Serena Common Tomcat installations. See the SctFoundLoc option.

For examples of the option settings needed for each database (Derby, Oracle, MySQL, and MS SQL Server), see [\(Windows\) DB Examples - Server Silent Install \[page 39\]](#).

(Windows) DB Examples - Server Silent Install

This section contains examples of Windows options files configured for each of the databases (Derby, MySQL, Oracle, and MS SQL Server).

Derby Database optionsFile.txt example

```
AgreeToLicense=Yes
SRA_USER_INSTALLDIR=C:\Documents and Settings\Administrator\.serena\ra
JMS_PORT=7918
AGENT_MUTUAL_AUTH=Y
TC_PORT=8080
DB_TYPE=derby

DERBY_PORT=11377

DERBY_USER=serena_ra

DERBY_PASSWORD=serena_ra
```

MySQL Database optionsFile.txt example

```
AgreeToLicense=Yes
SRA_USER_INSTALLDIR=C:\Documents and Settings\Administrator\.serena\ra
JMS_PORT=7918
AGENT_MUTUAL_AUTH=Y
TC_PORT=8080
DB_TYPE=mysql
JDBC_DRIVER_SOURCE=c:\TestArea\libloc\mysql-connector-java-5.1.22-bin.jar
MYSQL_JDBC_DRIVER=com.mysql.jdbc.Driver

MYSQL_DB_CONN=jdbc:mysql://localhost:3306/serena_ra

MYSQL_USER=serena_ra

MYSQL_PASSWORD=serena_ra
```

Oracle Database optionsFile.txt example

```
...
...
DB_TYPE=oracle
JDBC_DRIVER_SOURCE=c:\TestArea\libloc\oracle-connector-java-5.1.22-bin.jar
ORA_JDBC_DRIVER=oracle.jdbc.driver.OracleDriver

ORA_DB_CONN=jdbc:oracle:thin:@localhost:1521/serena_ra

ORA_DB_SCHEMA=serena_ra

ORA_PASSWORD=serena_ra
```

SQL Server Database optionsFile.txt example

```
...
DB_TYPE=sqlserver
SQLSVR_JDBC_DRIVER=com.microsoft.sqlserver.jdbc.SQLServerDriver
SQLSVR_JDBC_DRIVER=com.microsoft.sqlserver.jdbc.SQLServerDriver

SQLSVR_DB_CONN=jdbc:sqlserver://localhost:1433;DatabaseName=serena_ra
```



```
SQLSVR_USER=serena_ra
```

```
SQLSVR_PASSWORD=serena_ra
```

Server Silent Install (Linux/UNIX (AIX, Solaris))

NOTE

(Linux 64bit systems only) The ia32-libs package must be installed for 32bit compatability mode; the Serena Release Automation server installer is a 32bit application.

To install the server in silent mode:

1. Download the installation files. For more information, see [Downloading Serena Release Automation \[page 34\]](#).
2. Create an `optionsFile.txt` and save it to the root directory (see [\(Linux/UNIX \(AIX, Solaris\)\) Server Silent Install Options](#)).
3. In Linux/UNIX(AIX or Solaris), as a user with root privileges issue the command:

```
SerenaRA-server.bin -silent -options optionsFile.txt
```

Where:

`optionsFile.txt` is a file that contains the properties you have set for your system.

For examples of the `optionsFile.txt`, see [\(Linux/UNIX \(AIX, Solaris\)\) DB Examples - Server Silent Install \[page 42\]](#).

(Linux/UNIX (AIX, Solaris)) Server Silent Install Options

This section provides the list and description of the Linux/UNIX (AIX, Solaris) silent install options, and examples of the option settings needed for each database (Derby, Oracle, MySQL, and MS SQL Server).

For examples of the `optionsFile.txt`, see [\(Linux/UNIX \(AIX, Solaris\)\) Server Examples - DB Silent Install \[page 42\]](#).

(Linux/UNIX AIX, Solaris) Server silent install options table

Option	Default	Description
IS_SELECTED_INSTALLATION_TYPE_	typical	must be set to typical
SctFoundLoc		To install a new Serena Common Tomcat, leave value empty. (Required) When you want to reuse an existing Serena Common Tomcat; provide the path and also specify the <code>SctInstallLoc</code> option.
SctInstallLoc		Used with the <code>SctFoundLoc</code> option. (Required) When you want to reuse an existing Serena Common Tomcat; can be a new location or reinstall location.
DbDetailsVendor	derby	Use to specify a database vendor other than the default: oracle mysql sqlserver
DbDetailsUser	serena_ra	database user ID
DbDetailsPwd	password	password for database user ID
DbDetailsSchema		Required. Database schema
DbDetailsDriver		database driver class, for example: <code>org.apache.derby.jdbc.ClientDriver</code>

Option	Default	Description
dbDetailsDriverFilename	derby	Required for a database other than Derby; specify a database driver file.
DbDetailsDerbyPort	11377	Required for Derby; specify the Derby port.
DbDetailsConnection		database connection
ServerDetailsMutualAuth	false	agent mutual authentication option: true false
ServerDetailsPort	7918	server port
IS_INSTALL_MODE	silent	must be set to: silent
SctTomcatOwner		system user to own Serena common Tomcat files; valid for new Serena common Tomcat scenario.
SctTomcatPort	8080	Tomcat port. Only used for new Serena Common Tomcat installations. See the SctFoundLoc option [page 41] .

NOTE

The -V syntax is required.

(Linux/UNIX (AIX, Solaris)) DB Examples - Server Silent Install

This section contains sample Linux/UNIX (AIX, Solaris) optionsFile.txt files; each is configured for one of the databases (Derby, MySQL, Oracle, and MS SQL Server).

MySQL Database optionsFile.txt example

```
-V IS_SELECTED_INSTALLATION_TYPE=typical
-V SctFoundLoc=""
-V SctInstallLoc="/opt/serena/common"
-V DbDetailsVendor="mysql"
-V DbDetailsUser="sra"
-V DbDetailsPwd="sra"
-V DbDetailsDriver="com.mysql.jdbc.Driver"
-V dbDetailsDriverFilename=""
-V DbDetailsConnection="jdbc:mysql://localhost:3306/serena_ra"
-V ServerDetailsMutualAuth="false"
-V IS_INSTALL_MODE="silent"
-V SctTomcatOwner=dmsys
-V SctTomcatPort="8080"
```

Derby Database optionsFile.txt example

```
-V IS_SELECTED_INSTALLATION_TYPE="typical"
-V SctFoundLoc=""
-V SctInstallLoc="/opt/serena/common"
-V DbDetailsVendor="derby"
-V DbDetailsUser="User01"
-V DbDetailsPwd="MyPassword"
-V DbDetailsDriver="org.apache.derby.jdbc.ClientDriver"
-V dbDetailsDriverFilename=""
-V DbDetailsDerbyPort="11377"
-V DbDetailsConnection="jdbc:derby://localhost:11377/data"
-V ServerDetailsMutualAuth="false"
-V IS_INSTALL_MODE="silent"
-V SctTomcatOwner="dmsys"
-V SctTomcatPort="8080"
```

Oracle Database optionsFile.txt example

```

-V IS_SELECTED_INSTALLATION_TYPE="typical"
-V SctFoundLoc=""
-V SctInstallLoc="/opt/serena/common"
-V DbDetailsVendor="oracle"
-V DbDetailsUser="User02"
-V DbDetailsPwd="MyPassword"
-V DbDetailsDriver="oracle.jdbc.driver.OracleDriver"
-V dbDetailsDriverFilename=""
-V DetailsConnection="jdbc\:oracle\:thin\:@localhost\:1521"
-V ServerDetailsMutualAuth="false"
-V IS_INSTALL_MODE="silent"
-V SctTomcatOwner="dmsys"
-V SctTomcatPort="8080"

```

MS SQL Server Database optionsFile.txt example

```

-V IS_SELECTED_INSTALLATION_TYPE="typical"
-V SctFoundLoc=""
-V SctInstallLoc="/opt/serena/common"
-V DbDetailsVendor="sqlserver"
-V DbDetailsUser="User03"
-V DbDetailsPwd="MyPassword"
-V DbDetailsDriver="com.microsoft.sqlserver.jdbc.SQLServerDriver"
-V dbDetailsDriverFilename=""
-V DbDetailsConnection="jdbc\:sqlserver\://localhost\:1433;DatabaseName=serena_ra"
-V ServerDetailsMutualAuth="false"
-V IS_INSTALL_MODE="silent"
-V SctTomcatOwner="dmsys"
-V SctTomcatPort="8080"

```

Agent Installation

For production environments, Serena 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.

IMPORTANT!

Except for evaluation purposes, do *not* install an agent on the same machine as the server.

Each agent needs the appropriate rights to communicate with the Serena Release Automation server.

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 Serena Release Automation 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.

Interactive Agent Installation (Windows, Linux/UNIX (AIX, Solaris))

NOTE

(Linux 64-bit systems only) The ia32-libs package must be installed for 32-bit compatibility mode; the Serena Release Automation agent installer is a 32-bit application.

This type of installation utilizes a wizard that guides you through the complete process.

To install the agent:

1. Download the installation files. For more information, see [Downloading Serena Release Automation" \[page 34\]](#).
2. Run the downloaded installer `SerenaRA-Agent.exe`, or your platform-specific alternative, and follow the step-by-step instructions.

Silent Mode Agent Installation

This section contains information about how to implement a silent install of the Serena Release Automation agent on Windows, Linux/UNIX (AIX, Solaris).

IMPORTANT!

Before you can implement a silent install of an agent, you *must* create and save an `optionsFile.txt`.

(Windows) Agent Silent Installation

To install an agent in silent mode:

1. Download the appropriate agent installer for your platform. For more information, see [Downloading Serena Release Automation \[page 34\]](#).
2. Create an options file and save as:

```
C:/optionsFile.txt
```

3. In Windows, record a response file for a silent install as follows:

```
<agent installer> -r <filename>
```

4. Issue the command:

```
SerenaRA-Agent.exe -I silent -f optionsFile.txt
```

Where:

```
optionsFile.txt
```

is the options file you created in [Step 2](#).

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

Serena Release Automation 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 Serena 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.

Agent Relay Configuration table

Parameter	Description
Directory where you would like to install the agent relay	Enter the directory where you want the agent relay installed.
Java home	Directory where Java is installed. Ensure that the JAVA_HOME environment variable points to this directory.
Name of this relay	Enter the name of the agent relay. Each relay must have a unique name. The default name is <code>agent-relay</code> .
IP or hostname which this agent relay should use	Enter the IP or hostname on which the relay will listen.
Port which this agent relay should proxy HTTP requests on	Enter the port on which the agent relay should listen for HTTP requests coming from agents. The default value is 20080.
Port which this agent relay should use for communication.	Enter the port on which the agent relay will use for JMS-based communications with remote agents. The default value is 7916.
Connect the agent relay to a central server?	Specify whether you want the relay to connect to the Serena Release Automation server.
IP or hostname of your central server	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.
Port which the central server uses for communication	If you indicated that you want to connect the relay to the server, enter the port the server uses to communicate with agents. The default value is 7918.

Parameter	Description
Use secure communication between the agent, relay and server?	Specify whether you want to use SSL security for communications between server, relay, and remote agents. The default value is <code>Y</code> . IMPORTANT! 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).
Use mutual authentication between the agent, relay and server.	If mutual authentication is required, enter <code>Y</code> . See SSL Configuration [page 46] 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 <code>N</code> .

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 Serena Release Automation

You upgrade the Serena Release Automation server, agents, and agent relays independently. Before upgrading, download the appropriate installation package from the Serena support portal (upgrades are done with the same package used for installation), and uncompress it.

1. Run the installation script for the item you want to upgrade. To upgrade the server, for example, run the `install-server` script; to upgrade an agent, run the `install-agent` script.
2. When prompted for the location of the installation directory, enter the path to an existing installation. When you specify an existing installation, Serena Release Automation 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: server, agent, or agent relay.

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.

Serena Release Automation 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. Serena Release Automation 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!

Serena Release Automation automatically uses SSL in unauthenticated mode for JMS-based communications between the server and agents (JMS is Serena Release Automation'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. Serena Release Automation certificates are self-signed. When mutual authentication mode is active, Serena Release Automation uses it for JMS-based server, local agents, and agent relay communication.

To activate this mode, the Serena Release Automation 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 Serena 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.

Property Settings Required Before Exchanging Keys

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

Property	Location	Value
<code>server.jms.mutualAuth</code>	in the server's <code>server_install/conf/server/installed.properties</code> file	<code>true</code>
For each agent:		
<code>locked/agent.mutual_auth</code>	in the agent's <code>agent_install\conf\agent\installed.properties</code> file	<code>true</code>
For each agent relay:		
<code>agentrelay.jms_proxy.secure</code>	in the relay's <code>relay_install\conf\agentrelay.properties</code> file	<code>true</code>
For each agent relay:		
<code>agentrelay.jms_proxy.mutualAuth</code>	in the relay's <code>relay_install\conf\agentrelay.properties</code> file	<code>true</code>

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].crt
```

You should see the message:

```
Certificate stored in file agent_name.crt.
```

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].crt -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 Serena Release Automation

Both UNIX- and Windows-based installations require the Serena Release Automation 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 [Database Installation \[page 35\]](#).

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 Serena Release Automation 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 Serena Release Automation

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

User name: `admin`

Password: `admin`

You can change these later by using the **Settings** tab on the Serena Release Automation web application (see [System Settings \[page 139\]](#)).

3. Activate the license.

A license is required in order for the agents to connect to the server. Without a license, Serena Release Automation will be unable to run deployments. For information about acquiring and activating a license, see [Licenses \[page 140\]](#).

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 Serena Release Automation 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 Servers and Agents \[page 32\]](#) 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 \[page 67\]](#) 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 \[page 95\]](#) 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 Serena Release Automation 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.

helloWorld: Creating Components

Components contain the artifacts, such as files, images, and databases 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.

helloWorld: 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 and name it: `helloWorld`.
2. Within this `helloWorld` directory, create a sub-directory and name it `hello`.
3. Within the subdirectory create and save five text-type files and name them `artifact1.txt`, `artifact2.txt`, and so on.

These five files represent the artifacts that you will be deploying. You will be creating a component that contains these files.

NOTE

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

helloWorld: Component Version

1. Display the New Component dialog `Home > Components > New Component`

New Component dialog

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

- Name ***: A text input field.
- Description**: A text input field.
- Template**: A dropdown menu currently set to 'None'.
- Source Config Type**: A dropdown menu currently set to 'None'.
- Import Versions Automatically**: An unchecked checkbox.
- Copy to CodeStation**: A checked checkbox.
- Default Version Type ***: A dropdown menu currently set to 'Full'.
- Inherit Cleanup Settings**: A checked checkbox.

At the bottom right of the dialog are 'Save' and 'Cancel' buttons.

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.

For this exercise, ignore the Template field. Templates provide a way to reuse commonly used component configurations. For information about templates, see [Component Templates \[page 86\]](#).

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

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

Source Config Type



`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.

- 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).

Imported Artifacts

The screenshot shows the SERENA CodeStation interface. The breadcrumb navigation is `Home > Components > helloWorld > Versions > Version: hello`. The main heading is **Version: hello**. Below the heading are tabs for `Main`, `Edit`, and `Properties`. A table with columns `Status`, `Description`, `Created`, `By`, and `Actions` is shown, with a message: "No statuses have been assigned to this version." and an `Add a Status` button. Below this, a summary states **Total: 0.1 KB (6 files)**. A table lists the artifacts:

Name	Size	Last Modified	Version	Actions
artifact1.txt	24 bytes	10/5/12 10:13 AM	1	Download
artifact2.txt	24 bytes	10/5/12 10:13 AM	1	Download
artifact3.txt	24 bytes	10/5/12 10:13 AM	1	Download
artifact4.txt	24 bytes	10/5/12 10:13 AM	1	Download
artifact5.txt	24 bytes	10/5/12 10:13 AM	1	Download
artifact6.txt	24 bytes	10/5/12 10:13 AM	1	Download

helloWorld: 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:

- Display the Create New Process dialog for the `helloWorld` component `Home > Components > helloWorld > Processes > Create New Process`.

Create New Process dialog

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.

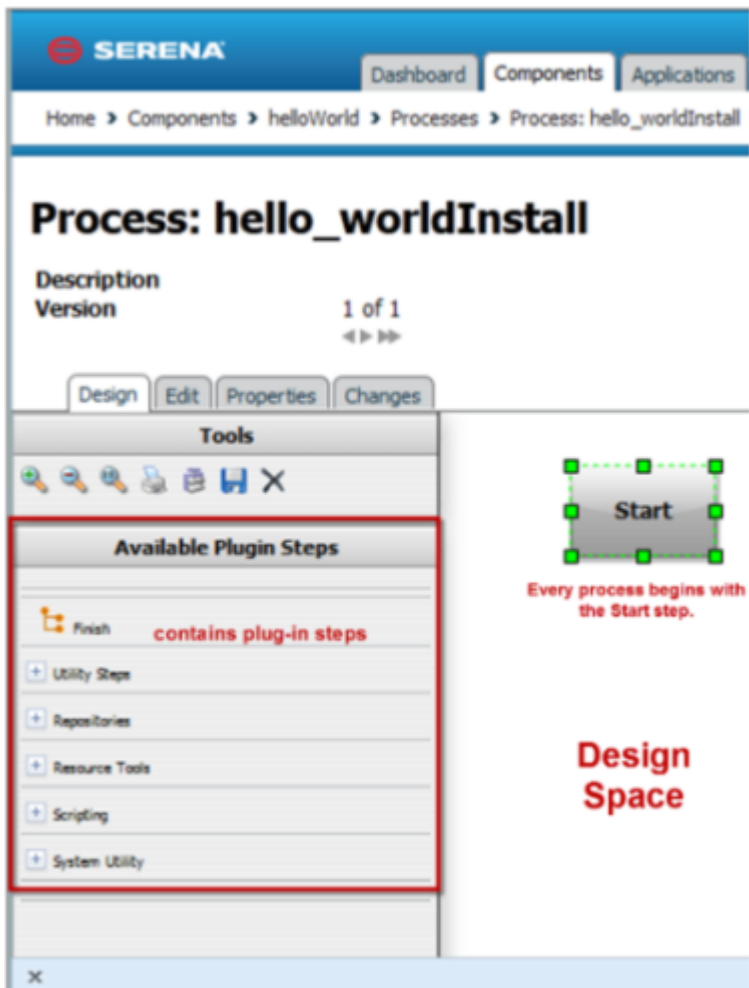
helloWorld Process Design

To complete the process you define 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`

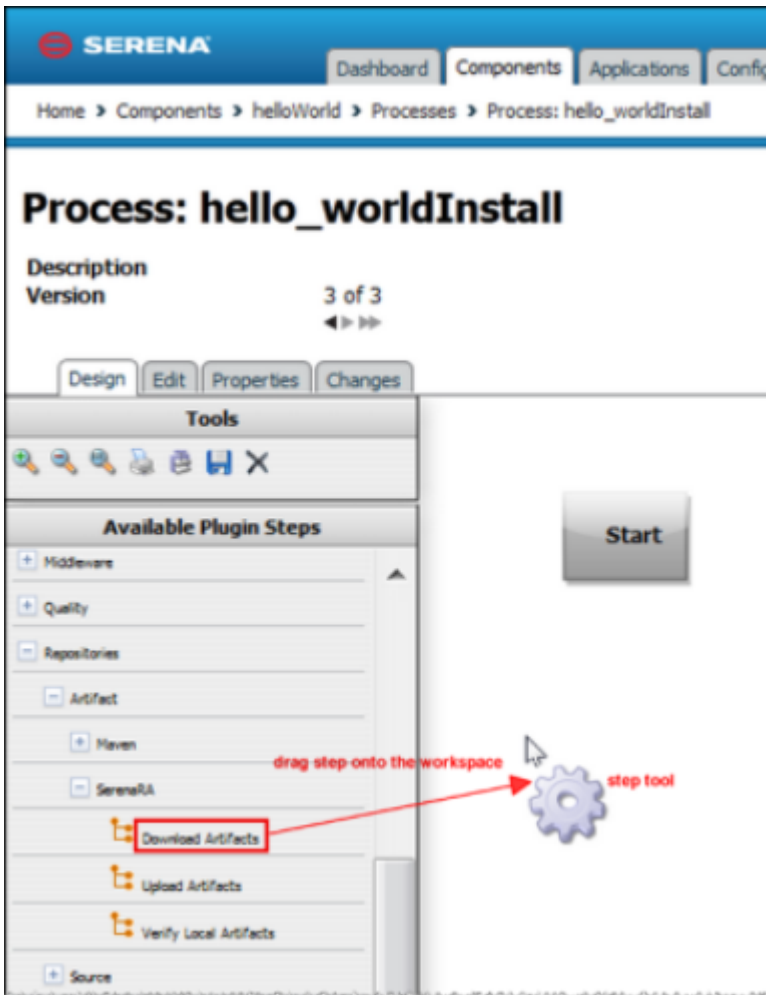
Process Design Pane



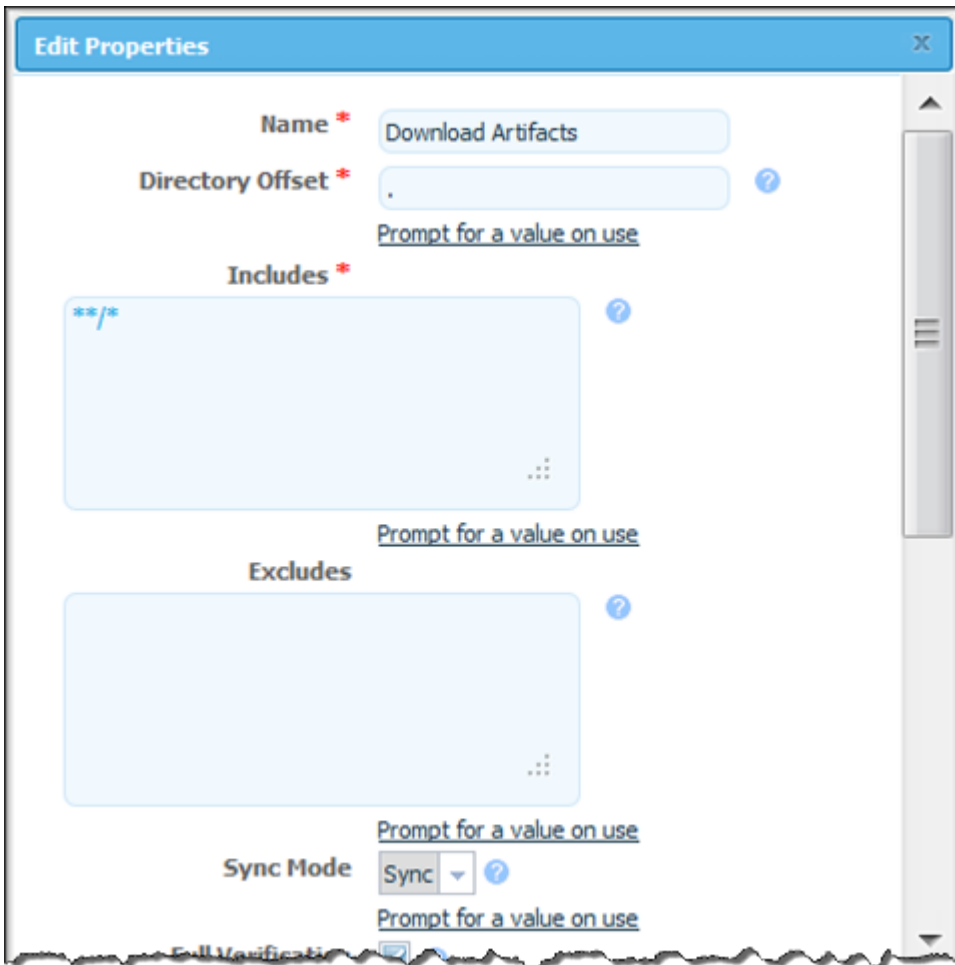
The steps are listed in the Available Plug-in Steps list-box. Take a moment to expand the listings and review the available steps. Many commonly used plug-in steps are available out-of-the-box.

2. In the Available Plug-in Steps box, expand the `Serena Release Automation` menu item `Repositories > Artifact > Serena Release Automation`.
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.

Adding a Step to the Process



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

Edit Properties Dialog

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 Serena Release Automation can create a directory during processing, you specify any target directory. I entered `c:\hello` which did not exist on my system, and let Serena Release Automation create it for me.

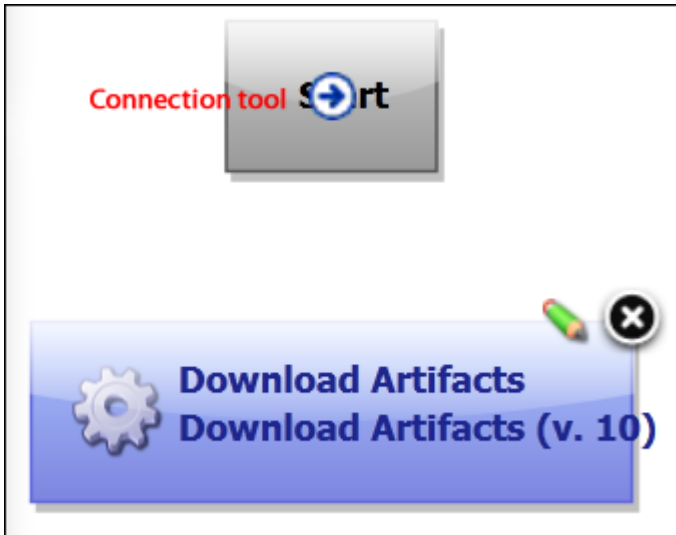
If the field is left blank, the process will use the working directory defined earlier. Entering the path overrides the previous value and will cause the source files to be moved—deployed—to the entered location when the process runs. The default value would move (download) the files to `agent_directory\work\component_name_directory`.

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

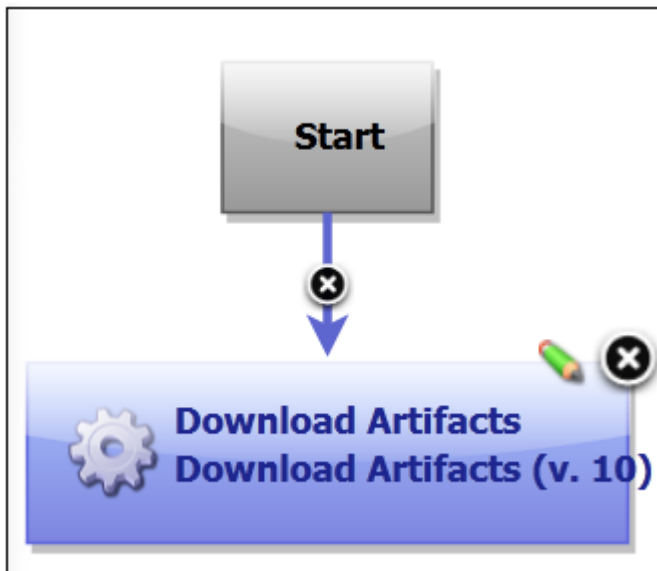
5. Next, the step must be positioned within the process workflow. There's no requirement that a step be positioned immaculately after it's created; you could place several more before defining their positions, but because this is the only step we are adding, it makes sense to define its position now.

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

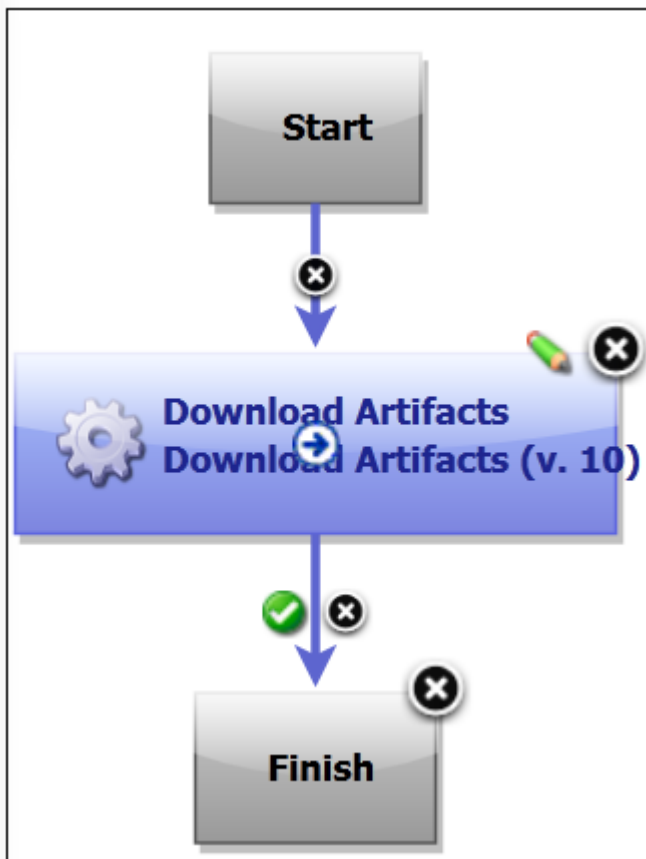
Hover the mouse pointer over the `Start` step to display the connection tool, as shown in the following illustration. Each step has a connection tool which is used to connect it to other steps.

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.

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.

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 Serena Release Automation, 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.

helloWorld: Creating an Application

To create an application:

1. Display the Create New Application dialog `Home > Applications > Create New Application [button]`.
2. Enter a name, for example, `helloWorld_application`, and an (optional) description.

There are no naming requirements. However the number of associated items, for example components, processes, applications, and environments can become large; so, we recommend you use a naming scheme that makes it easy to identify related items.

3. For the Notification Scheme, accept the default value of `None` from the drop-down, and save the application.

Serena Release Automation 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) when a deployment finishes. Notifications can also play a role in deployment approvals (see [Notifications \[page 144\]](#)).

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.

helloWorld: 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`.

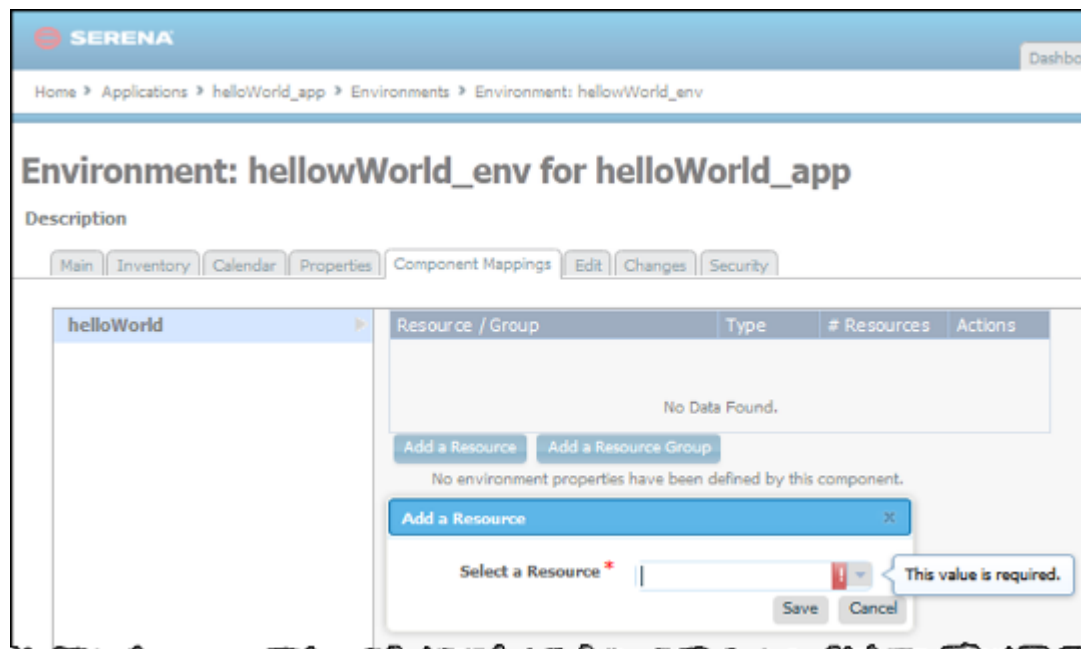
Create New Environment

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 \[page 110\]](#) for information about the approval process. This is our first deployment so an uncontrolled environment will do fine—leave the box unchecked.
 - 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.
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 Serena Release Automation 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 \[page 90\]](#) for information about resource groups.

Component Mappings



helloWorld: 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:

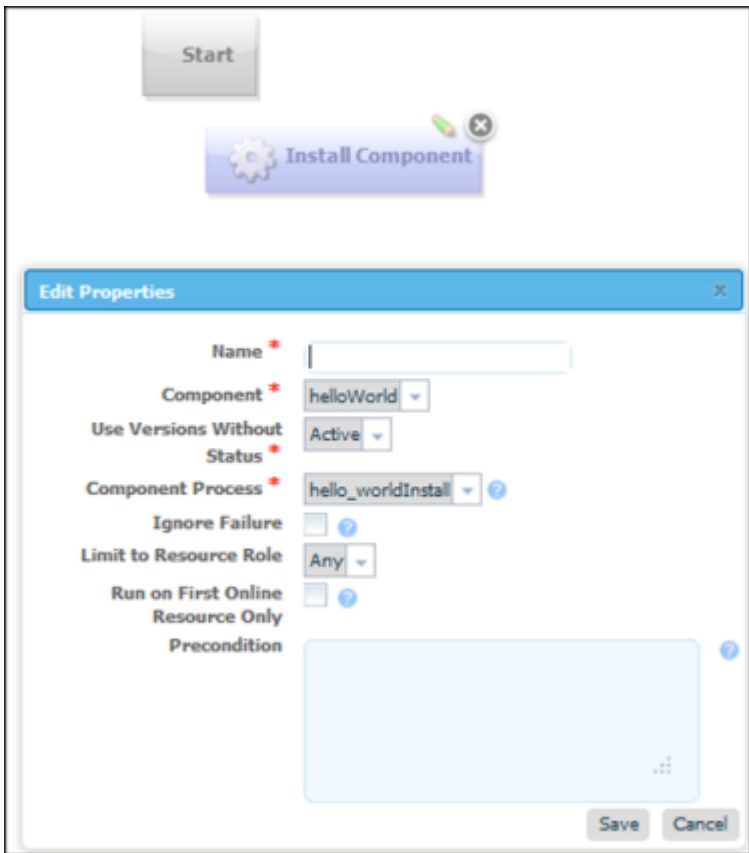
- The Required Application Role drop-down field is used to restrict who can run this process. The default value, None, means anyone can run the process. The available options are derived from the Serena Release Automation Security System. For information about security roles, see [Serena Release Automation Security \[page 128\]](#)
- 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 [Inventory \[page 150\]](#), 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 when you used the Process Design pane to create the helloWorld component process (see [Component Processes \[page 74\]](#)).

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.

Edit Properties dialog

Select a component from the **Component** drop-down list box. If you followed the *Quick Start*, 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 \[page 95\]](#) for information about the other fields), and click **Save**.

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

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 can run the application.

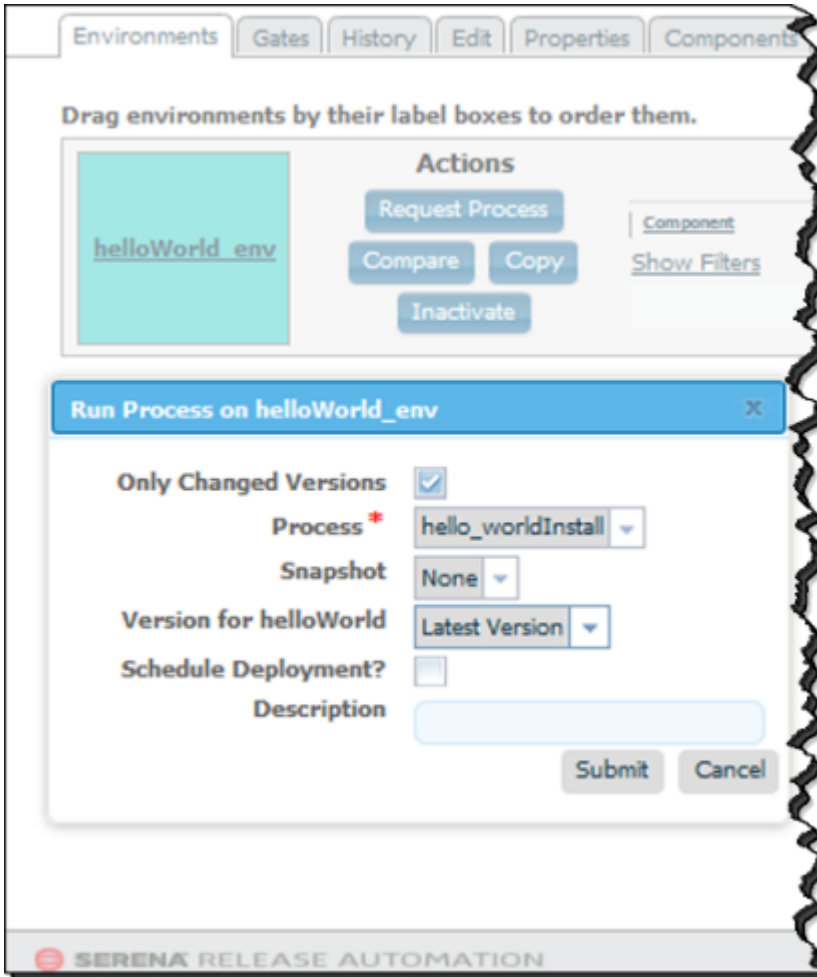
1. On the Application pane, click the Request Process button for the environment you created earlier.

Request Process



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

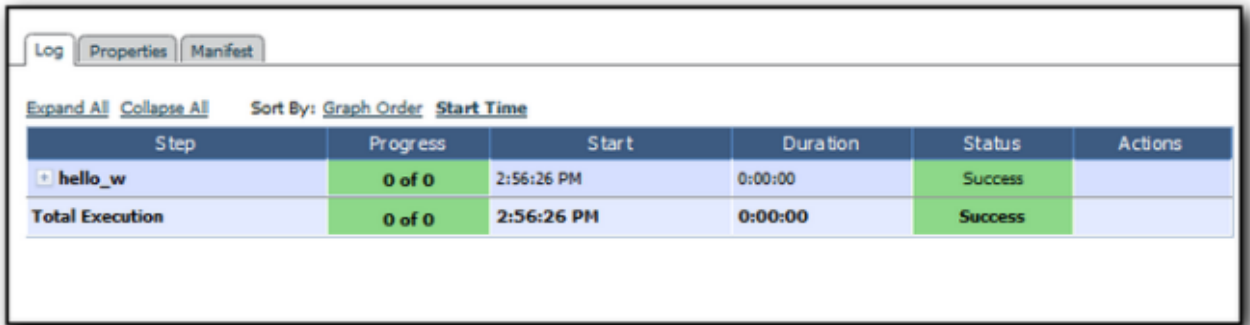
Run Process Dialog



3. Click Submit to run the application.

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

Application Process Request



The screenshot shows a web interface for an application process request. At the top, there are three tabs: 'Log', 'Properties', and 'Manifest'. Below the tabs, there are links for 'Expand All' and 'Collapse All', and a 'Sort By' dropdown menu currently set to 'Graph Order' with 'Start Time' as an alternative option. The main content is a table with the following data:

Step	Progress	Start	Duration	Status	Actions
<input type="checkbox"/> hello_w	0 of 0	2:56:26 PM	0:00:00	Success	
Total Execution	0 of 0	2:56:26 PM	0:00:00	Success	

Take a few moments to examine the information on this pane; hopefully, you will see a Success message.

4. To see additional information (Output Log, Error Log, Application Properties), click the Details link.

Using Serena Release Automation

This documentation contains the following information about using Serena Release Automation:

- [Components \[page 67\]](#)
- [Resources \[page 90\]](#)
- [Agents \[page 93\]](#)
- [Applications \[page 95\]](#)
- [Deployments \[page 110\]](#)
- [Reports \[page 114\]](#)

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 Serena Release Automation.

Component Versions and the CodeStation Repository

After defining a component's source and processes, you import its artifacts into Serena Release Automation'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 Serena Release Automation, 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 Serena Release Automation'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. Serena Release Automation 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 \[page 155\]](#).

Additionally, you can create processes and configure properties and save them as templates to create new components. See [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 Serena Release Automation does with the component's artifacts. A process might consist of any number of steps, such as starting and stopping 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 [Component Templates \[page 86\]](#)), or you can import a component directly (see [Importing/Exporting Components \[page 87\]](#)).

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.
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.

Fields Available for All Component Source Types table

Field	Description
Name	Identifies the component; appears in many UI features. Required.
Description	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.
Template	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 <code>None</code> . 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 Component Templates [page 86] .
Source Config Type	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 Component Source Configuration [page 153]) 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.
Import Versions Automatically	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.
Copy to CodeStation	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. Serena recommends that the box be checked.

NOTE

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.

Field	Description
Default Version Type	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.
Inherit Cleanup Settings	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 \[page 153\]](#) 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 Serena Release Automation 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 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]`.
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 [Component Template Properties \[page 88\]](#)). 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 [Component Properties table \[page 70\]](#).

Component Properties table

Property Type	Description
Properties	Custom property; can be used in scripts and plug-ins. Properties inherited from templates cannot be modified on the component level.

Property Type	Description
Environment	<p>Available to environments that use the component. The property will appear on the environment's Component Mappings pane <code>Applications > [selected application] > Environments > [selected environment] > Component Mappings</code>, see Application Environments [page 96].</p> <p>Each property must have a type:</p> <ul style="list-style-type: none"> • <i>Text</i> Enables users to enter text characters. • <i>Text Area</i> Enables users to enter an arbitrary amount of text, limited to limited to 4064 characters. • <i>Check Box</i> Displays a check box. If checked, a value of <code>true</code> will be used; otherwise the property is not set. • <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. NOTE Not currently implemented. • <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. • <i>Secure</i> Used for passwords. Similar to <i>Text</i> except values are redacted.
Version	<p>Available to every component version <code>Components > [selected component] > Versions > [selected version] > Properties</code>. Values can be set at the individual version level. Each property must have a type (described above).</p>

Component Versions

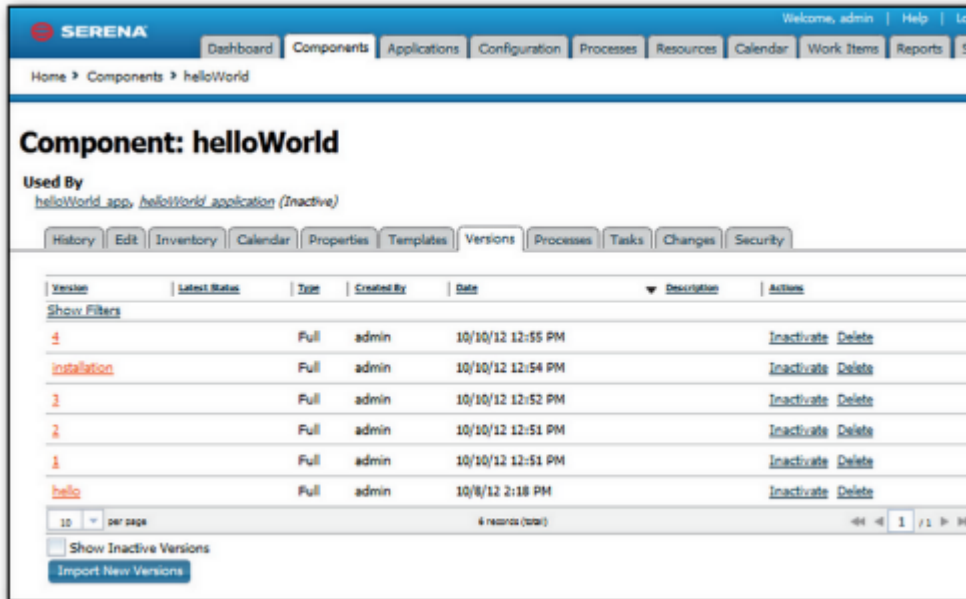
Each time a component's artifacts are imported into the repository, including the first time, it is versioned. Versions can be assigned automatically by Serena Release Automation, applied manually, or come from a build server. Every time a component's artifacts are modified and reimported, a new version 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`.

Component Version Pane



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.

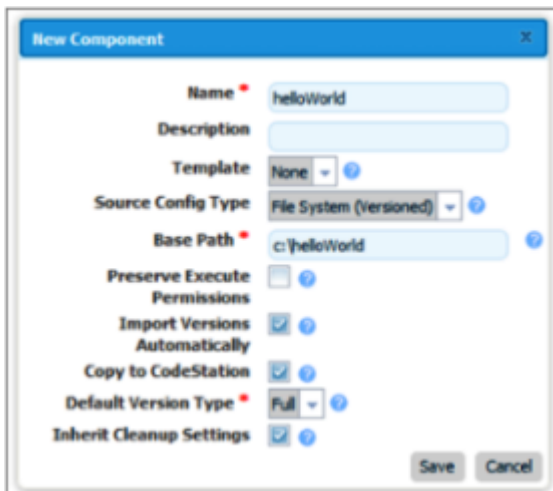
The optional description can be used to convey additional information about the component. If the component is used by more than one application, for example, entering "Used in applications A and B" can help identify how the component is used. If you are unsure about what to enter, leave the field blank. 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 [Component Templates \[page 86\]](#).

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

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

Source Config Type



`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 [System Settings \[page 139\]](#).

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 [Application Gates \[page 107\]](#)).

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.

Serena Release Automation-provided statuses are defined in an XML file which you can freely edit to add your own values. See [Structure of the default.xml File \[page 109\]](#).

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 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.

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 Serena Release Automation'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 [Process Editor \[page 75\]](#)

Serena Release Automation 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, Serena Release Automation provides plug-ins for many common processes, such as downloading and uploading artifacts, and retrieving environment information. See [Plug-ins \[page 155\]](#).

A frequently used process can be saved as a template and applied to other components. See [Component Templates \[page 86\]](#).

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

Create New Process Fields table

Field	Description
Name	Identifies the process; appears in many UI elements. Required.

Field	Description
Description	The optional description can be used to convey additional information about the process.
Process Type	<p>Required. Defines the process type. Available values are:</p> <ul style="list-style-type: none"> • Deployment: deploys a component version to the target resource and updates the inventory after a successful execution. • Configuration Deployment: 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. • Uninstall: standard uninstall that removes a component version from a target resource and the resource's inventory. • Operational (With Version): 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. • Operational (No Version Needed): same as the previous type, but does not require a component version.
Inventory Status	<p>Required. 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.</p>
Default Working Directory	<p>Required. Defines the location used by the agent running the process (for temporary files, etc.). The default value resolves to <code>agent_directory\work\component_name_directory</code>. 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.</p>
Role	<p>Required. Restricts who can run the process. The available options are derived from the Serena Release Component Automation security system. The default value is <i>None</i>, meaning anyone can run the process. For information about security roles, see Serena Release Automation Security [page 128].</p>

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

Process Editor

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

Displaying the Process Editor

To access the process editor:

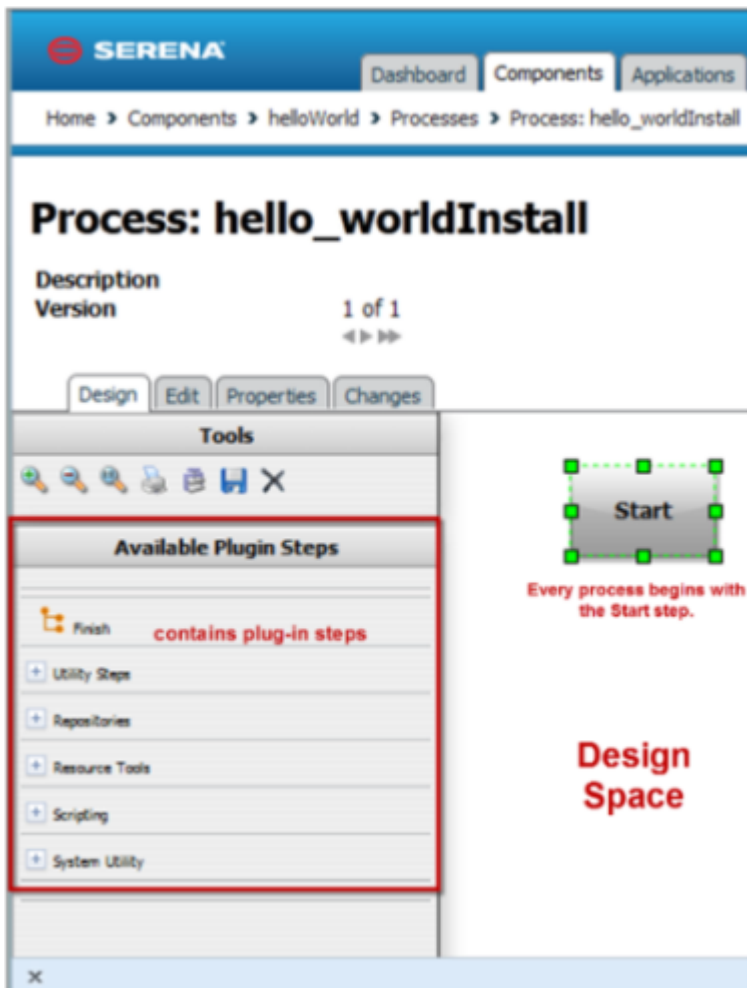
1. On the **Component: name** pane, click the **Processes** tab.
2. Click on the name of the process you want to edit.

Component Processes



The **Process Design** pane is displayed.

Process Design Pane



Available steps are listed in the **Available Plug-in Steps** list. Serena Release Automation provides several utility steps and plug-ins which are highlighted in [Process Design Pane \[page 77\]](#). This figure also shows several user-installed plug-ins.

Using the Process Editor

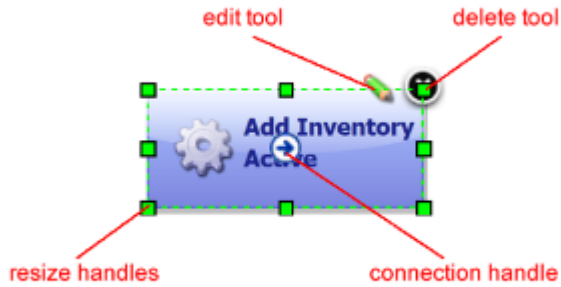
When the **Process Design** pane opens, the **Design** view displays. Processes are assembled on the **Design** view. Several other views can be displayed by clicking the associated tab:

Available Views table

Tab/View	Description
Edit	Displays the Edit view where you can change process parameters. See Component Processes [page 74] .
Properties	Displays the Properties view where you can create and change process properties.
Changes	Displays the Process Changes view. This view provides a record for every process change—property add or delete, and process save or delete.

Processes are assembled by dragging individual steps onto the design space, configuring, and then connecting them. When a step is dragged onto the design space, a pop-up displays that is used to configure the step. Once configured and the pop-up closed, you define relationships between steps by dragging *connection handles* between associated steps.

Typical Process Step



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

Anatomy of a Step table

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

NOTE

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

Adding Process Steps

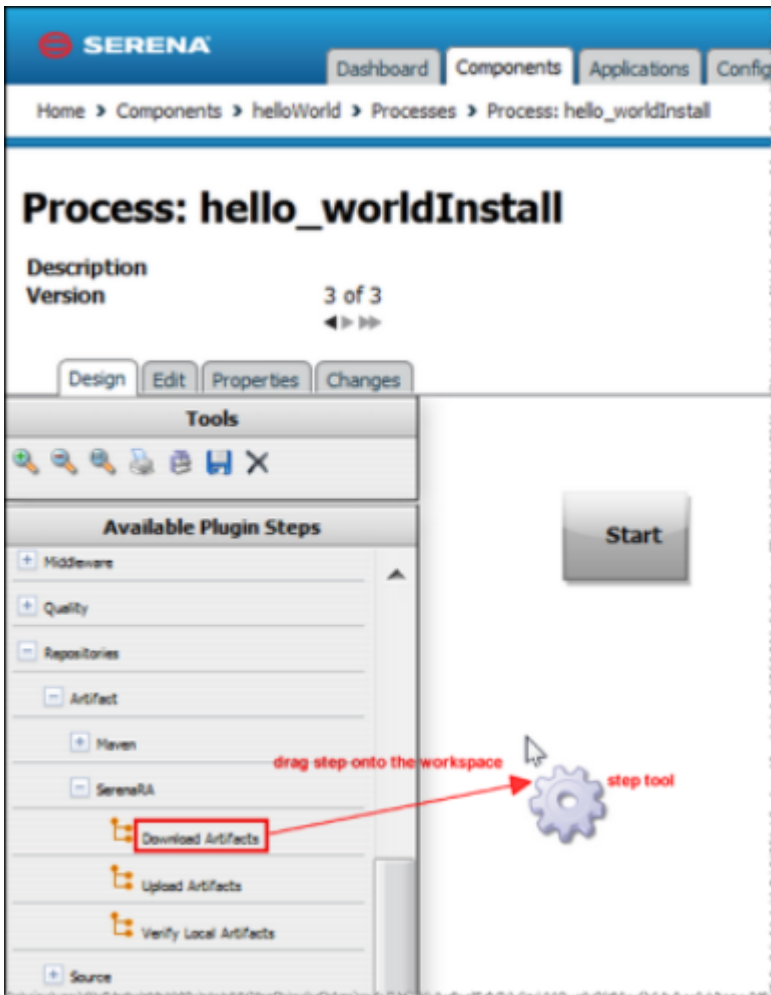
To add a step:

1. In the **Available Plug-in Steps** list, using your mouse drag the step you want to use onto the design space.

The cursor changes to the *step tool*.

2. Release the step tool over the design space.

Adding a Step



The **Edit Properties** pop-up displays.

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 \[page 155\]](#) for information about configuring specific steps.

Typical Edit Properties Pop-up

The screenshot shows the 'Edit Properties' dialog box for a step named 'Download Artifacts'. The dialog is titled 'Edit Properties' and has a close button (X) in the top right corner. The fields and their values are as follows:

- Name ***: Download Artifacts
- Directory Offset ***: .
- Includes ***: **/*
- Excludes**: (empty)
- Sync Mode**: Sync
- Full Verification**:
- Set File Execute Bits**:

Each field has a 'Prompt for a value on use' link and a help icon (question mark) to its right. The dialog also features a vertical scrollbar on the right side.

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

3. After configuring the step's properties, to save the step click **Save**.

The step is on the design space and ready to be connected to other steps.

4. If you want to remove the step from the design space, click **Cancel**.

You can add connections immediately after placing a step, or place multiple 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; 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 your mouse pointer over the step that you want to use as the connection's origin.

The pointer changes to the connection tool.

Connection Tool

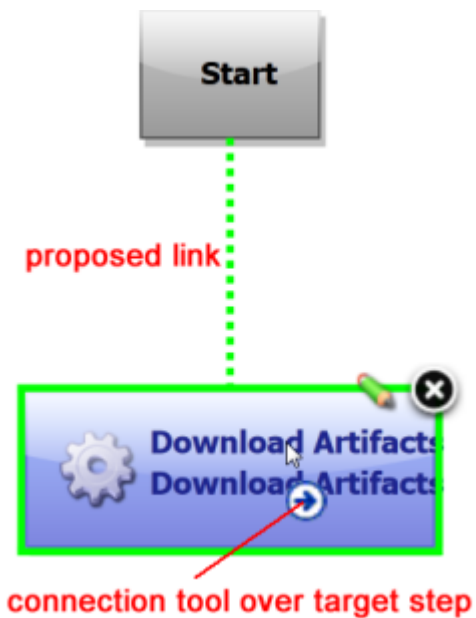
connection tool



2. Drag the connection tool over the target step.

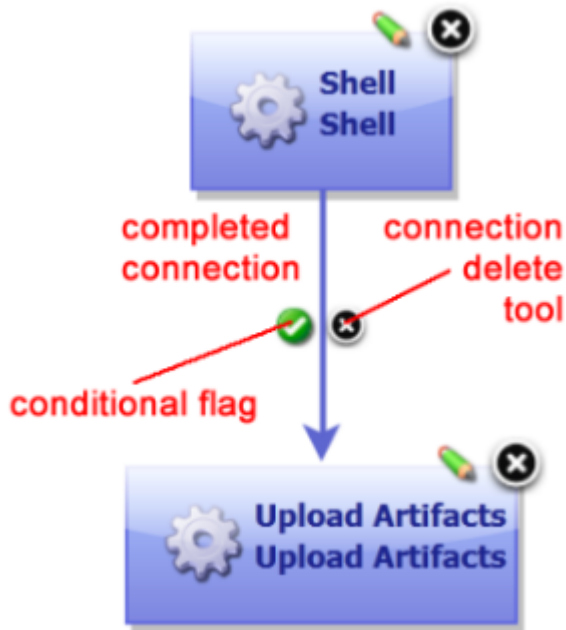
When the target step is highlighted, release the mouse to create a connection.

Dragging the Connection Over a Target Step



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

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.

Edit Properties Dialog

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 a step that uses it (see [Component Processes \[page 74\]](#)).

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.

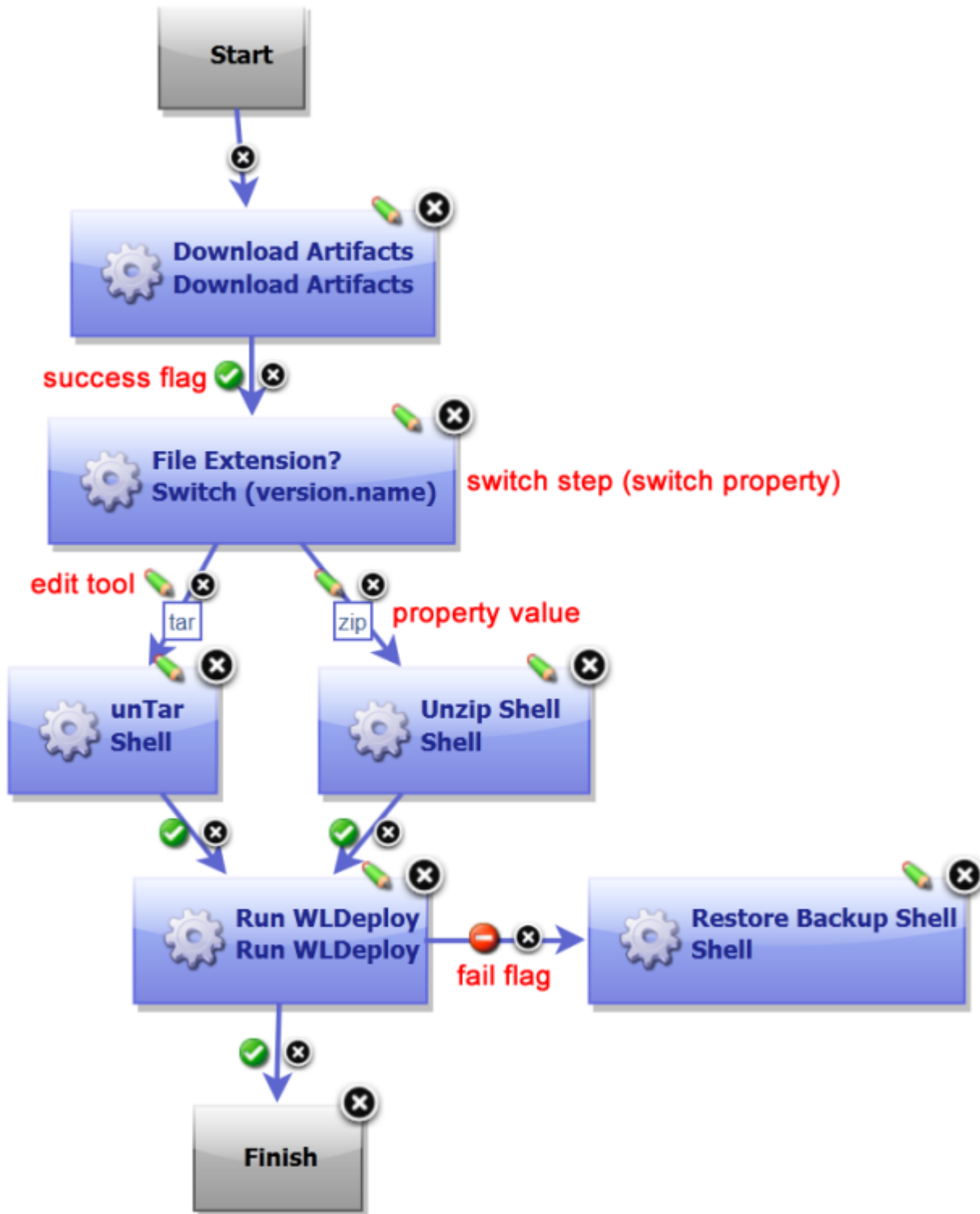
both

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.

Process with Switch Step



A *switch step* is a utility step supplied by Serena Release Automation 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

`Run WLDeploy` has success and fail conditions. See [Plug-ins \[page 155\]](#) 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.

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 [Work Items \[page 106\]](#)). 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 [Notifications \[page 144\]](#)). If a scheme is not specified, the default scheme is used. The available notification schemes 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.

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 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. For information about notification schemes, see [Notifications \[page 144\]](#), and for information about creating component processes see [Process Editor \[page 75\]](#).

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 [Creating Plug-ins \[page 158\]](#).

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 [<post-processing> Element \[page 163\]](#) for more information.

Component Templates

There are two types of templates available:

component template

Enables you save and reuse component processes and properties and create new components from them; template-based components inherit the template properties and process(es).

configuration template

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]`.
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 Serena Release Automation-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 \[page 153\]](#)).

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 [Component Processes \[page 74\]](#).

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.

Component Template Properties table

Template Property Type	Description
Properties	<p>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.</p> <p>Every component will have this property; it will appear on the Create New Component dialog for every component created from this template (see Creating Components [page 67]). A value defined here can be changed by created components. Each property must have a type:</p> <ul style="list-style-type: none"> • <code>Text</code> Enables users to enter text characters. • <code>Text Area</code> Enables users to enter an arbitrary amount of text, limited to limited to 4064 characters. • <code>Check Box</code> Displays a check box. If checked, a value of <code>true</code> will be used; otherwise the property is not set. • <code>Select</code> Requires a list of one or more values which will be displayed in a drop-down list box. Enables a single selection. <p>NOTES</p> <p>Not currently implemented.</p> <ul style="list-style-type: none"> • <code>Multi Select</code> Requires a list of one or more values which will be displayed in a drop-down list box. Enables multiple selections. • <code>Secure</code> Used for passwords. Similar to <code>Text</code> except values are redacted.
Component Property Definitions	

Template Property Type	Description
	<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 Applications > [<i>selected application</i>] > Environments > [<i>selected environment</i>] > Component Mappings), see Application Environments [page 96]. A value defined here can be changed by environment. Each property must have a type:</p> <ul style="list-style-type: none"> <li data-bbox="256 415 347 436">• Text <p data-bbox="288 470 759 491">Enables users to enter text characters.</p> <li data-bbox="256 527 419 548">• Text Area <p data-bbox="288 581 1369 602">Enables users to enter an arbitrary amount of text, limited to limited to 4064 characters.</p> <li data-bbox="256 638 419 659">• Check Box
Environment Property Definitions	<p data-bbox="288 695 1466 716">Displays a check box. If checked, a value of <code>true</code> will be used; otherwise the property is not set.</p> <ul style="list-style-type: none"> <li data-bbox="256 751 376 772">• Select <p data-bbox="288 806 1445 869">Requires a list of one or more values which will be displayed in a drop-down list box. Enables a single selection.</p> <p data-bbox="288 898 379 919">NOTES</p> <p data-bbox="288 953 624 974">Not currently implemented.</p> <li data-bbox="256 1010 464 1031">• Multi Select <p data-bbox="288 1064 1422 1127">Requires a list of one or more values which will be displayed in a drop-down list box. Enables multiple selections.</p> <li data-bbox="256 1163 376 1184">• Secure <p data-bbox="288 1197 1066 1218">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 any process(es) the template may have (see [Component Processes \[page 74\]](#)), and any possible properties (see [Component Properties \[page 70\]](#)).

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 [Creating Components \[page 67\]](#).

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 contain configuration data. Typically, the data is for server configurations, for example on Tomcat servers, 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]`.
2. Enter a 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 Serena Release Automation. In general, 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.

Deleting and Deactivating Components

Components can be deactivated or 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 is removed—along with all versions—from the database and CodeStation and cannot be activated at a later time. The original artifacts are not affected; only the CodeStation copies are deleted.

NOTE

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, Serena Release Automation 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 Serena Release Automation 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

Serena Release Automation 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

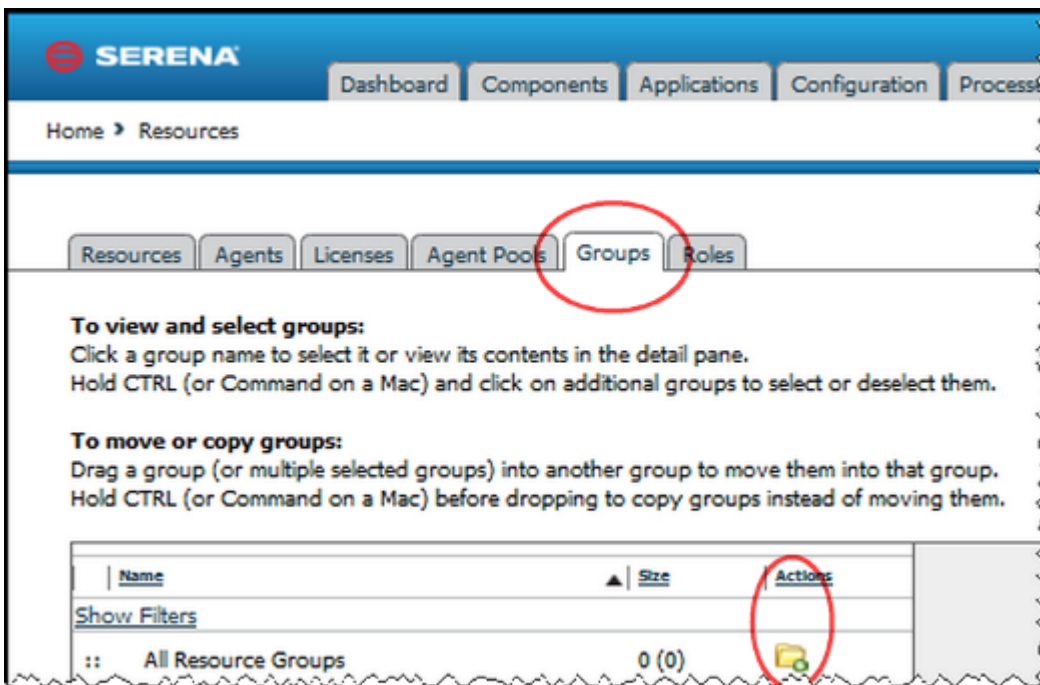
Before continuing, ensure that at least one agent has been installed in a target environment (for evaluation purposes, the agent can be on the same machine as the server).

Creating a Resource Group

To create a resource group:

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

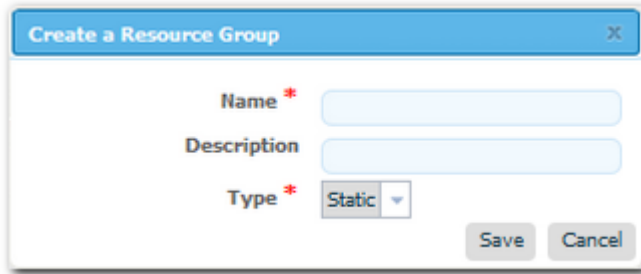
Action Tool



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

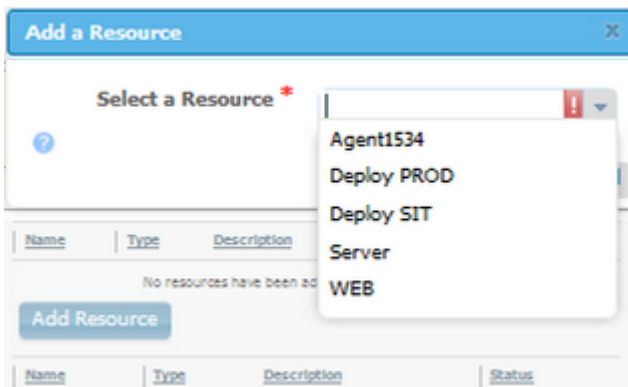
Name and description. Typically, the name will correspond to either the Environment the Resource participates in, the Component that uses the Resource Group, or a combination of both, for example SIT, DB, or SIT-DB. What description you give depends on how you intend to use the Resource to which this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

Create a Resource Group Dialog



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

Add a Resource Dialog



- 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 Group creation.

Sub-resources

Name	Size	Actions
Show Filters		
☰ All Resource Groups	1 (0)	📁
QA	0 (0)	✎ 📁 📁
☰ SIT	1 (1)	✎ 📁 📁
APP	1 (1)	✎ 📁 📁
WEB	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 Serena Release Automation for rolling deployments, balancing, and such. 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 for example, 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. When you add the role to a resource, you can set the values for the properties. For example, if you create a `WS` role and define a `serverURL` property for it, you can access the property like this:

```
${p:resource/WS/serverURL}
```

For information about Serena Release Automation properties, see [Serena Release Automation Properties \[page 164\]](#).

Agents

An agent is a lightweight process that runs on a target host and communicates with the Serena Release Automation 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 Serena Release Automation'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 Serena Release Automation 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 Serena Release Automation 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:

Remote Agent Installation fields table

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.

Field	Description
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.
Server Host*	Host name or IP address of the Serena Release Automation server or agent relay to which the agent will connect.
Server Port*	Serena Release Automation 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 Serena Release Automation 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.

Agent Management table

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 <code>Offline</code> .
Upgrade	This option will shutdown the agent and apply the upgrade. While the agent is shutdown, its status will be <code>Offline</code> . After the upgrade is applied, the agent will be restarted. Before its status is <code>Online</code> , it might briefly be <code>Connected</code> .
Test	This option will perform an agent settings and connection test. Test results are displayed in the Connection Test dialog.
Inactivate	This option will deactivate the agent. Agents that are deactivated cannot perform deployments. To reactivate the agent, check the <code>Show Inactive Agents</code> check box on the Agents pane, then click <code>Activate</code> 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. Optional. 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.

Agent Pool Management table

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, and such. These are called *application processes*. However, application level processes are only concerned with the components and resources necessary for deployment — differentiating application processes from component processes (which are concerned with running commands, and so on).

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 \[page 90\]](#)).

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.

Serena Release Automation maintains an inventory of every artifact deployed to each environment and tracks the differences between them.

Application Processes

Application processes, like component processes, are created with the process editor. Serena Release Automation 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. Serena Release Automation tracks the history of each component version, which enables application processes to restore environments to any desired point.

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, Serena Release Automation 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, Serena Release Automation 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 [Importing/Exporting Applications \[page 98\]](#) for information about importing applications).

After creating an application, you:

- add components ([Adding Components to an Application \[page 97\]](#))
- create an environment ([Creating an Environment \[page 100\]](#))
- associate an agent with the environment ([Mapping Resources to an Environment](#))
- create an application process ([Application Processes \[page 96\]](#))

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 \[page 90\]](#).

To create an application:

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

New Application Information table

Field	Description
Name and Description	Typically, correspond to the application you plan on deploying.
Notification Scheme	notifications—based on events—can be sent out due to Serena Release Automation integrations with LDAP and e-mail servers. For example, when an application deployment fails or succeeds, the default notification scheme sends out an email. Notifications can also be used to send out emails to a user or a group (based on their security role) for approval of a requested deployment (see Notifications [page 144]).
Enforce Complete Snapshots	If selected, the application requires every component to get versioned.

2. Save your work when done.

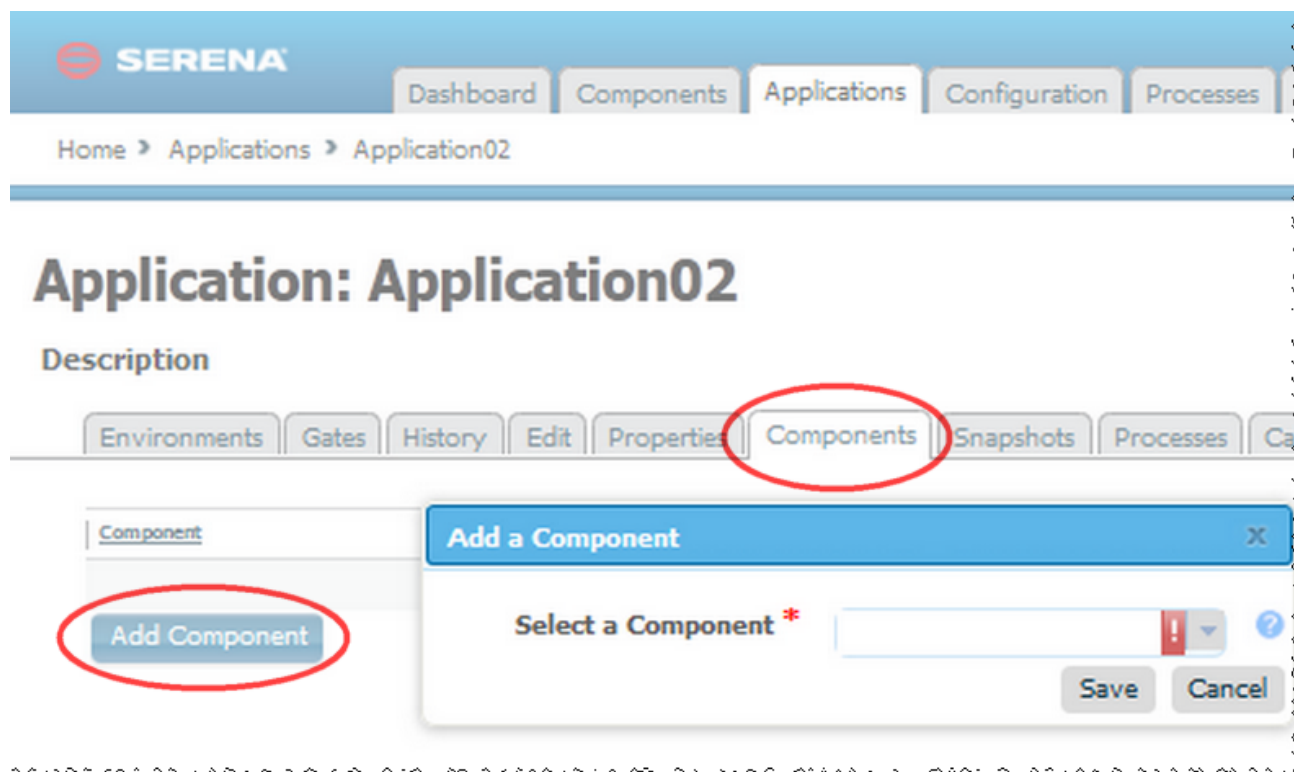
Adding Components to an Application

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]`

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 Serena Release Automation 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), Actions field, click the `Export` link. 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 [Importing/Exporting Components \[page 69\]](#).

NOTE

If imported components have the Import Versions Automatically parameter set to true, Serena Release Automation 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.

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]`.
2. Enter the environment information in the fields provided:

New Application Information table

Field	Description
Name and Description	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, you may want to use the name "SIT". Description is optional text.
Require Approvals	To require an approval before components can be deployed to the environment, select this check box. If checked, Serena Release Automation 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.
Lock Snapshots	If you want all snapshots used in this environment to be locked to prevent changes, select the check box.
Color	Select a color to visually identify the environment in the user interface.
Inherit Cleanup Settings	Determines how many and for how long component versions are kept in CodeStation. By default, this check box is selected; the application will use the values specified on the System Settings pane. If unchecked, additional fields display: the Days to Keep Versions (initially set to -1, keep indefinitely) and Number of Versions to Keep (initially set to -1, keep all), which enable you to define custom values.

3. Click Save.

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 \[page 90\]](#).
2. Display the Component Mappings pane `Applications > [selected application] > Environments > [selected environment] > Component Mappings`.

Component Mapping

The screenshot shows the SERENA web interface. At the top, there is a navigation bar with tabs for Dashboard, Components, Applications, Configuration, and Profile. Below this is a breadcrumb trail: Home > Applications > App 123 > Environments > Environment: Dev62. The main heading is 'Environment: Dev62 for App 123'. Underneath is a 'Description' section with a sub-navigation bar containing tabs for Main, Inventory, Calendar, Properties, Component Mappings (circled in red), Edit, and Approval Process. The main content area is split into two panes. The left pane shows a component named 'helloWorld'. The right pane is titled 'Resource / Group' and lists three resource groups: 'Deploy PROD', 'QA', and 'WEB'. At the bottom of the right pane are two buttons: 'Add a Resource' and 'Add a Resource Group'. Below the buttons, it says 'No environment properties have been d'.

3. 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).
4. 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 [Resource Groups \[page 91\]](#).
 - 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}`

Creating Application Processes

To create an application process:

1. Display the Create an Application Process dialog `Applications > [select application] > Create New Process [button]`.
2. Enter the following information:

Application Process Fields table

Field	Description
Name/Description	Typically the name and description correspond to the application you plan on deploying.
Required Application Role	Use this drop-down to select the role a user must have in order to run the application. For information about creating application roles, see Roles and Permissions [page 129] . The default value is <code>None</code> . If you want to handle inventory manually, select <code>Advanced</code> .
Inventory Management	To have inventory handled automatically, leave the default value, <code>Automatic</code> , selected.

Specify how the process reacts if expected agents are offline:

Check Before Execution

Checks to see if expected agents are on line before running the process. If agents are off line, the process will not run.

Offline Agent Handling

Use All Available; Report Failure:

Process will run as long as at 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.

Always Report Success

Process will run as long as at least one agent defined in the environment is on line; reports successful deployments.

3. Save your work.

Application process—the steps comprising them—are configured with the process editor. For information about using the process editor, see [Process Editor \[page 75\]](#). For information about individual process steps, see [Application Process Steps](#).

Application Process Steps

Application processes, like component processes, are created with the [process editor \[page 75\]](#). Serena Release Automation provides several common process steps, otherwise application processes are assembled from processes defined for their associated components.

Application Process Step Details

The application process steps are described in the following topics:

- [Finish \[page 103\]](#)

- [Install Component \[page 103\]](#)
- [Uninstall Component \[page 103\]](#)
- [Rollback Component \[page 104\]](#)
- [Manual Application Task \(Utility\) \[page 104\]](#)

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.

Install Component Properties table

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 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 Serena Release Automation. 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.

Uninstall Component Properties table

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.

Field	Description
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 Serena Release Automation. 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.

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 Serena Release Automation. 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 [Work Items \[page 106\]](#)). 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 \[page 105\]](#).

Manual Application Task Properties table

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 [Notifications \[page 144\]](#).

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 [Work Items \[page 106\]](#)). 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 [Notifications \[page 144\]](#)). 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 \[page 105\]](#). 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: [Environ_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 [Process Editor \[page 75\]](#).

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.

Creating Snapshots

To create a snapshot:

1. Display the New Application Snapshot pane Home > Application > Snapshots > Create New Snapshot.

2. Enter the name of your snapshot in the Name field.
3. In the Process Version Locking field, specify how you want Serena Release Automation to select component processes:

Always use Latest Version

Default. Use the most recently defined component process version for each component in the application.

Lock to Current Versions

Use the current component process version for each component.

4. 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.

5. 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`.

Version statuses are defined in the `default.xml` file which you can freely edit to add your own values, see [Structure of the default.xml File \[page 109\]](#). 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`.

Gates Pane

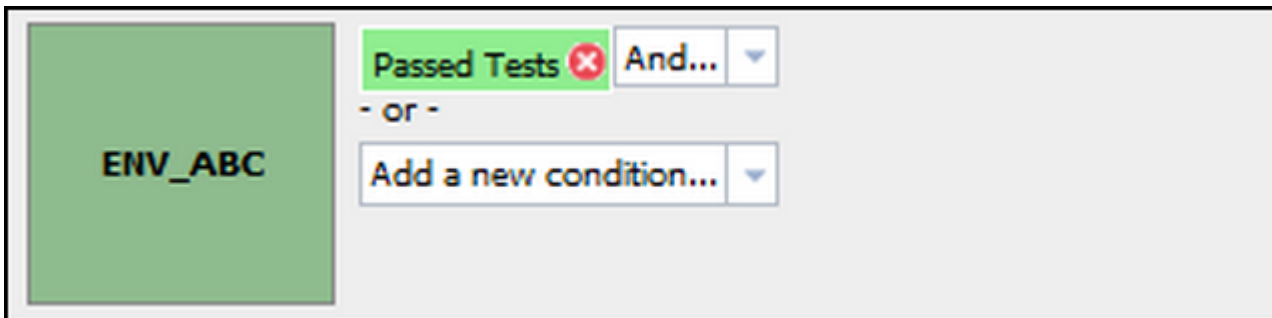
The screenshot shows the Serena Release Automation interface for 'App 123'. The 'Gates' pane is active, displaying three environment cards: 'ENV_ABC' (green), 'Dev62' (orange), and 'Production' (purple). Each card has an 'Add a new condition...' dropdown menu. The dropdown for 'ENV_ABC' is open, showing options: 'Add a new condition...', 'Add a new condition...', 'Archived', 'Latest', and 'Passed Tests'. A 'Save Conditions' button is located at the bottom left of the gate configuration area.

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.

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 [Component Version Statuses \[page 73\]](#) for more information about component statuses.

Structure of the default.xml File

Inventory and versions statuses are defined in the `default.xml` file. You can modify the supplied values for both types as well as add your own values. If you modify the file, restart the server in order to see your changes.

Here an example of `default.xml`:

```
<?xml version="1.0"?>
<status-scheme name="Default">
<inventory-statuses>
<status name="Active" color="#8DD889" unique="true" />
<status name="Staged" color="#80D8FF" />
</inventory-statuses>
<version-statuses>
<status name="Latest" color="#F6F4D8" unique="true" />
<status name="Passed Tests" componentRoleName="StatusAdder" color="#FFDDAA" />
<status name="Archived" color="#AAAAAA" />
</version-statuses>
</status-scheme>
```

Each `<status/>` element has a required `name` attribute and several optional ones, defined in the following table:

<status> Attributes table

Attribute	Description
name	Identifies the status; appears in user-interface. Used to create gates, and available in process steps.

Attribute	Description
color	Hexadecimal color definition; determines the color displayed in the user interface.
unique	Boolean value (true false). Only one component version with this status/attribute will be deployed to the environment.
componentRoleName	Security role required by user to add this status to the component version.

All attributes must be enclosed in double-quotes.

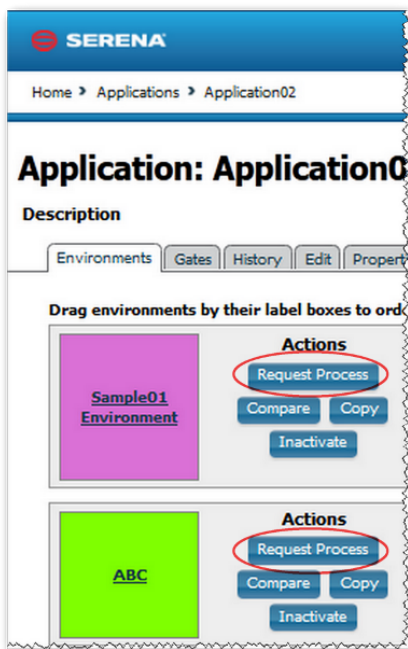
NOTE

While you can add as many values as you like, you cannot create new status types (only inventory- and version- statuses are supported). Additionally, all values must be defined in `default.xml`.

Deployments

Deployments are done with applications (see [Creating Applications \[page 97\]](#) 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`.

Request Process Actions

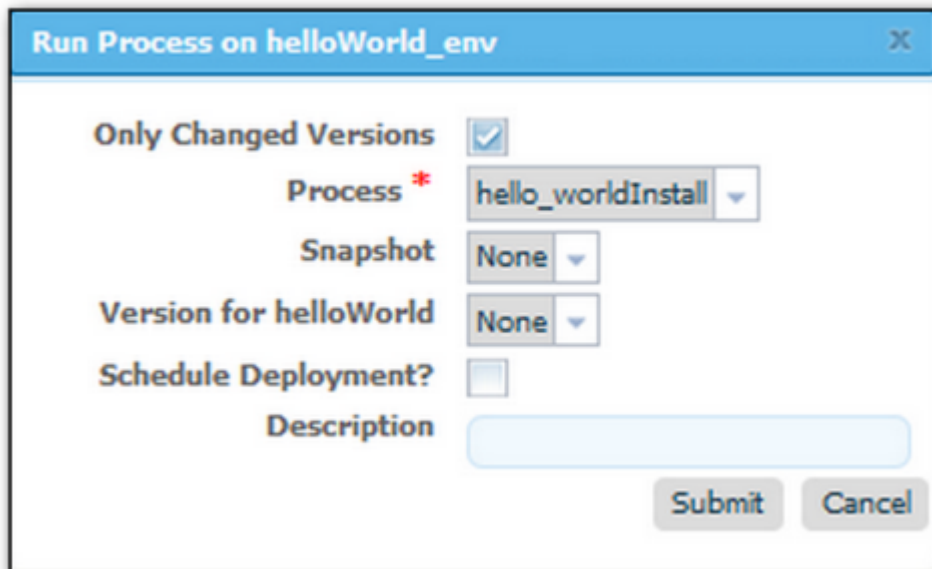


To run an application:

1. Go to the Applications tab.
2. Click the application.
3. For the environment where you want to perform the deployment, under **Actions** click **Request Process**.
The Run Process dialog displays.
4. Click the **Run Process** link for the environment you want to use.

The Run Process dialog displays.

Run Process Dialog



5. 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 [Snapshots \[page 15\]](#).
6. 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:

Version Options table

Version Option	Description
None	No version for this component. Useful when performing multi-component deployments or testing.
Specific Version	Enables you select any version already in Codestation.
Latest Version	Automatically uses the most recently imported version.
Latest With Status	Automatically uses the most recently imported version with the specified status. Status values are: Latest (default value), Passed Test, Archived.
All With Status	All component versions with the selected status will be deployed. Status values are: Latest, Passed Test, Archived.
All in Environment	All component versions already deployed in the environment with the selected status will be deployed. Status values are: Active (default), Staged.
All in Environment (Reversed)	All component versions already deployed in the environment with the selected status will be deployed in reverse order. Status values are: Active (default), Staged.

7. 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).
8. Select the process you want to run from the Process list box. All processes created for the application are listed.

9. 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.
10. 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.

Application Process Request Pane

The screenshot displays the 'Application Process Request' pane for 'helloWorld_app'. The interface includes a navigation bar with 'SERENA' and user information 'Welcome, admin | Help | About | Log Out'. The main content area shows the process details:

- Process:** [hello_worldInstall \(Version 3\)](#)
- Environment:** [helloWorld_env](#)
- Only Changed Versions:** true
- Date Requested:** 10/22/12 1:57 PM
- Requested By:** admin
- Scheduled For:** 10/22/12 1:57 PM
- Description:** [View Deployment Request](#)







Below the details are buttons for 'Log', 'Properties', and 'Manifest'. A table below shows the execution steps:

Step	Progress	Start	Duration	Status	Actions
hello_w	0 of 0	1:57:11 PM	0:00:00	Success	
Manual Task: Environment Approval		1:57:11 PM	0:00:07	Success	Completed by admin
Total Execution	0 of 0	1:57:11 PM	0:00:07	Success	

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

Deployment of Component Pane

The screenshot shows the SERENA web interface. The top navigation bar includes 'Welcome, admin | Help | About | Log Out' and a menu with 'Dashboard', 'Components', 'Applications', 'Configuration', 'Processes', 'Resources', 'Calendar', 'Work Items', 'Reports', and 'Settings'. The breadcrumb trail is 'Home > Components > helloWorld > Process Request on QA'. The main heading is 'Deployment of Component: helloWorld'. Below this, deployment details are listed: Process (hello_worldInstall (Version 8)), Version (2), Resource (QA), Agent (Agent1613m), Date (10/18/12 1:58 PM), and Requested By (admin). A link 'View Application Process Execution' is provided. Below the details are 'Log' and 'Properties' buttons. A table shows the execution steps, sorted by 'Start Time'.

Step	Type	Start	Duration	Status	Actions
Set Status: Success		1:58:01 PM	0:00:00	Success	
Create File	File Utils v.	1:58:01 PM	0:00:01	Success	  
Shell	Shell v.	1:58:02 PM	0:00:01	Success	  
Total Execution		1:58:01 PM	0:00:02	Success	

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

Scheduling Deployments

Serena Release Automation 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 > Request Process`. If you are scheduling a Snapshot deployment, go to `Application > Snapshots > Request Process` instead. Regardless of the type of deployment you are scheduling, configuration is the same.

After you check the Schedule Deployment box, Serena Release Automation 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. Serena Release Automation will prompt you to give the dates and times for the blackout.

Reports

Serena Release Automation 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 [Deployment Reports \[page 114\]](#)).

Security reports

Provide information about user roles and privileges (see [Security Reports \[page 124\]](#)).

For information about saving and printing reports (see [Saving and Printing Reports \[page 126\]](#)).

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

Deployment Reports table

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 Deployment Detail Report [page 114]).
Deployment Average Duration	Average deployment times for applications executed during a user-specified reporting period (see Deployment Average Duration Report [page 120]).
Deployment Total Duration	Total deployment times for applications executed during a user-specified reporting period (see Deployment Total Duration Report [page 122]).
Deployment Count	Provides information about the number of deployments executed during a user-specified reporting period (see Deployment Count Report [page 117]).

Security Reports table

Report	Description
Application Security	Provides information about user roles and privileges defined for Serena Release Automation-managed applications (see Application Security Report [page 125]).
Component Security	Information about user roles and privileges defined for components (see Component Security Report [page 125]).
Environment Security	Information about user roles and privileges defined for environments (see Environment Security Report [page 125]).
Resource Security	Information about user roles and privileges defined for resources (see Resource Security Report [page 126]).

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 [Saving and Printing Reports \[page 126\]](#).

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 [Saving and Printing Reports \[page 126\]](#).

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.

Deployment Detail Fields table

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.
	Final disposition of the deployment. Possible values are:
	<ul style="list-style-type: none"> • Success • Failure
Status	<ul style="list-style-type: none"> • Running • Scheduled • Approval Rejected • Awaiting Approval
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 View Request link displays the Application Process Request pane (see Applications [page 95]).

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.

Date Range table

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 Custom 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.

Report Filters table

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 Application 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 [Saving and Printing Reports \[page 126\]](#).

Report Samples: Deployment Detail

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

Sample Reports table

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

Field	Description
	<ul style="list-style-type: none"> • Plugin: Select the value from the drop-down list box. • Date Range: Use the Custom pop-up to set the start- and end-dates to the previous day.
Show me: <i>My deployments that used a specific application during the past month.</i>	<ul style="list-style-type: none"> • Application: Select the value from the drop-down list box. • User: Select your user ID. • Date Range: Select <code>Prior Month</code>.

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 fields

Deployment Count Fields table

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 Count 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`.

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 Custom 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.

Filters table

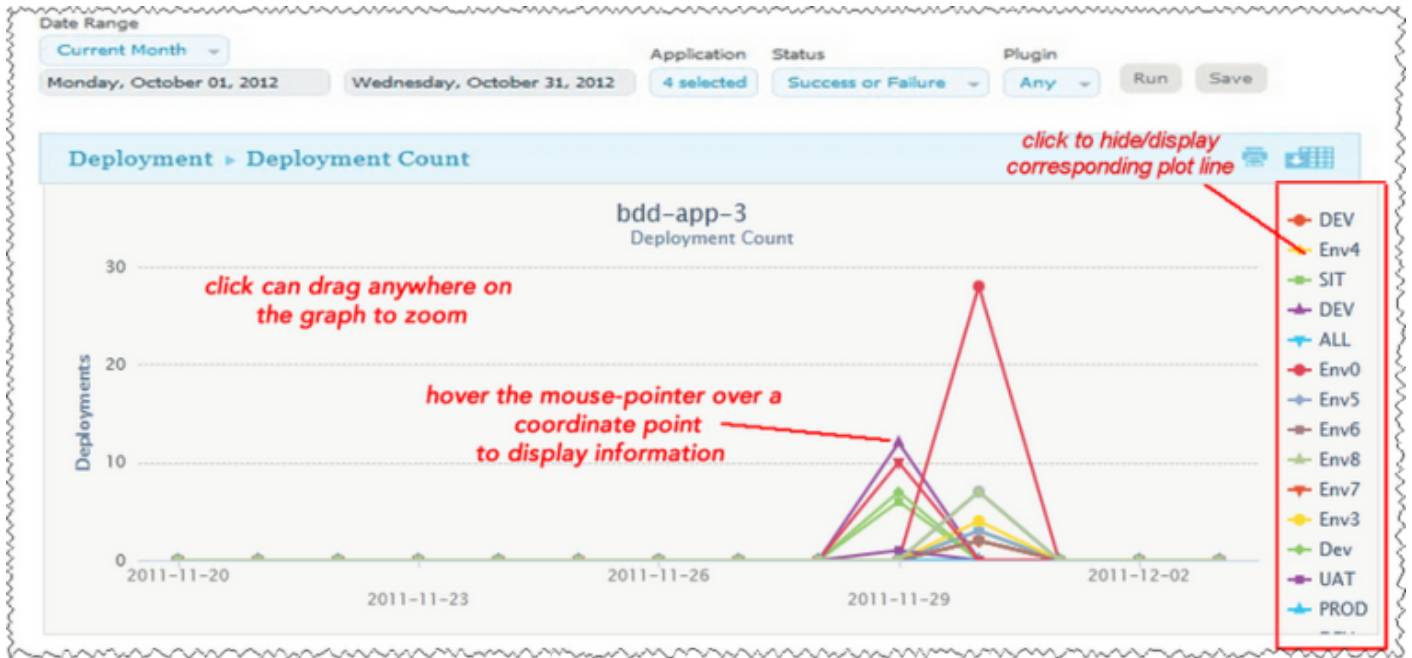
Field	Description
Application	<p>Only deployments executed by the selected application(s) appear in the report. To select applications:</p> <ol style="list-style-type: none"> 1. Click Application. 2. To include an application in the report, click the corresponding check box. If a large number of applications 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. 3. Click OK.
Status	Only deployments with the selected status appear in the report. Default value: <code>Success</code> or <code>Failure</code> , which means all deployments.
Plugin	Only deployments that used the selected plug-in appear in the report. Default value: <code>Any</code> . Note: the <code>Any</code> value also includes deployments that did not use a plug-in.

3. Run the report.

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

A tabular report and line graph are produced.

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 [Saving and Printing Reports \[page 126\]](#).

Report Samples: Deployment Count

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

Sample Reports

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

Field	Description
	<ul style="list-style-type: none"> • Date Range: Select <code>Prior Month</code>.
	<ul style="list-style-type: none"> • Application: Select the applications from the Applications dialog.
	<ul style="list-style-type: none"> • Status: <code>Failure</code>
	<ul style="list-style-type: none"> • Plugin: Select the value from the drop-down list box.
	<ul style="list-style-type: none"> • Date Range: Use the Custom pop-up to select the previous day.

Show me: *The number of failed deployments that used a specific plug-in yesterday.*

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

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`.

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 Custom 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.

Filters table

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"> 1. Click the Application button. 2. To include an application in the report, click the corresponding check box. <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"> 3. Click OK.
Status	<p>Only deployments with the selected status appear in the report. Default value: <i>Success</i> or <i>Failure</i>, which means all deployments.</p> <p>Only deployments that used the selected plug-in appear in the report. Default value: <i>Any</i>.</p>
Plugin	<p>NOTE</p> <p><i>Any</i> value also includes deployments that did not use a plug-in.</p>

3. Run the report.

Click **Run** 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 [Saving and Printing Reports \[page 126\]](#).

Sample Reports: Deployment Average Duration

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

Sample Reports table

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

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

Total Duration Fields table

Field	Description
Application	Name of the application that executed the deployment.

Field	Description
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`.

Date Range table

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 Custom 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.

Filters table

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"> 1. Click the Application button. 2. 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. 3. Click OK.
Status	Only deployments with the selected status appear in the report. Default value: <code>Success</code> or <code>Failure</code> , which means all deployments.

Field	Description
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 [Saving and Printing Reports \[page 126\]](#).

Sample Reports: Deployment Total Duration

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

Sample Reports table

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

Security Reports

Security Reports provide information about user roles and privileges defined with the Serena Release Automation security system.

Application Security Report

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

Application Security Fields

Application Security Fields table

Field	Description
Application	Name of the application.
Run Component Processes	Users who have component process execution privileges. For information about component processes, see Creating Components [page 67] .
Execute	Users who have application execution privileges. For information about applications, see Applications [page 95] .
Security	Users who can define privileges for other users. For information about security, see Serena Release Automation Security [page 128] .
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 [Saving and Printing Reports \[page 126\]](#).

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

Component Security Fields table

Field	Description
Component	Name of the component.
Execute	Users who have component process execution privileges. For information about component processes, see Creating Components [page 67] .
Security	Users who can define privileges for other users. For information about security, see Serena Release Automation Security [page 128] .
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 the column header.

For information about saving and printing reports, see [Saving and Printing Reports \[page 126\]](#).

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

Environment Security Fields table

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 Applications [page 95] .
Security	Users who can define privileges for other users. For information about security, see Serena Release Automation Security [page 128] .
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 the column header.

For information about saving and printing reports, see [Saving and Printing Reports \[page 126\]](#).

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

Resource Security Fields table

Field	Description
Resource	Name of the resource.
Execute	Users who have execution privileges for the resource. For information about resources, see Resources [page 90] .
Security	Users who can define privileges for other users. For information about security, see Serena Release Automation Security [page 128] .
Read	Users who can review information about the resource but not change it.
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 [Saving and Printing Reports \[page 126\]](#).

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

Serena Release Automation 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 **CSV**.
 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 from the Reports pane > My Reports menu.

To save a report:

1. Set the filter conditions.
2. Define the reporting period.
3. Run the report.
4. Optional. Set the sort order.

You can change the sort order for any column by clicking the column header.

5. Optional. Change column visibility. Click **Edit** 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 **Save**. The **Save Current Filters** dialog displays.
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 **Delete**.

Printing Reports**To print a report:**

1. Set the filter conditions.
2. Define the reporting period.
3. Run the report.
4. Optional. Set the sort order.

Your changes are reflected in the printed report.

5. Optional. 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 **Print** to print your report.

Administration

This documentation contains the following Administration sections:

- [Security \[page 128\]](#)
- [System Settings \[page 139\]](#)
- [Configuration \[page 146\]](#)
- [Inventory \[page 150\]](#)

Serena Release Automation Security

Serena Release Automation provides a flexible, [role-based security model \[page 129\]](#) that maps to your organizational structure. Different product areas, such as components, can be secured by roles. Each area has a set of permissions available to it. To configure security for an area, you create roles using the available permissions—execute, read, write, and so forth.

So, how are permissions applied to users? First, global default permissions can be granted. Default permissions are granted by product area and apply to all users. If default permissions are granted for, say, the agent area, a user will have those permissions even if she is also part of a group or role that does not.

Groups

Another way users can be granted permissions is by being a member of a group. Groups can have default permissions that apply to all group members. If a user is assigned to a group with default permissions for the agent area, as above, she will have those permissions even if she is also assigned a role that does not have them.

Roles and Permissions

Finally, users can be assigned to roles. Role members inherit a role's permissions. Except for UI and system security, users are assigned to roles on an item by item basis. For example, a user can be assigned a role that enables them to see only one application or only one component. Both groups and individual users can be assigned to roles.

Roles and permissions, including default permissions, are configured on an area by area basis; granting the execute permission to one role does not grant it to another. The default admin role has all permissions, but you can create another user with all permissions by creating a role for each area with all permissions granted, then assigning the user to each role. Typically, new roles are added to product areas during setup and occasionally thereafter.

While any number of roles can be created for an area, areas themselves cannot be created, modified (the available pool of permissions cannot be changed), or deleted.

Related Topics

- [Setting up Security \[page 128\]](#)

Setting up Security

Generally, you perform the following steps in order when setting-up Serena Release Automation security:

1. **Create Roles.**

Create roles and define permissions for the various product areas (see [Creating and Editing Roles \[page 130\]](#)). For most evaluations, the default roles should be adequate.

Use the UI security area to quickly assign access permissions to the different areas of Serena Release Automation.

Use the system security area to assign usage permissions, including the ability to define security for other users.

2. Authorization Realms.

Authorization realms are used by authentication realms to associate users with groups and to determine user access (see [Authorization Realms \[page 134\]](#)). Serena Release Automation includes both an internal database for storing security information as well as integration with the Lightweight Directory Access Protocol (LDAP). LDAP is a widely-used protocol for accessing distributed directory information over IP networks. If you are implementing a production version of Serena Release Automation, the LDAP integration is recommended. If you are evaluating Serena Release Automation, it is not necessary to set up the LDAP integration—full security is configured and enforced by the server.

3. Create Groups and Define Default Permissions.

Determine default permissions by product area. Global default permissions can be granted.

4. Create Authentication Realm.

The authentication realm is used to determine a user's identity within an authorization realm. If more than one realm has been configured, user authentication is determined following the hierarchy of realms defined on the Authentication pane. When a user attempts to log in, all realms are polled for matching credentials.

5. Add Users.

Add users to an authentication realm, then assign them to groups and roles. If you are using LDAP, you can import users and map them to the security system.

Roles and Permissions

Roles provide the building blocks for the security system. Roles have permissions that define the actions the roles can perform with product features. Typical actions include changing or executing an item, such as an application process, or modifying its security settings. Users or groups assigned to a role are automatically granted the permissions configured for it. The default roles can be edited and new roles can be created.

Serena Release Automation maps key product features or areas to security roles. Each area has several permissions defined for it (listed below). When you create a role, you first specify the product area. Selecting a product area defines the set of permissions available to the new role—only permissions defined for the area are available.

Generally, permissions fall into one of these groups:

Common Permissions table

Permission	Description
Security	Enables users to change an item's security settings. For example, a user with this permission for agents can determine which users can view, configure, and set security for them.
Write	Enables users to add, change, and delete items. A user with this permission for components can create a component.
Read	Enables users to read (view) an item, but not change it or create another of its type. A user with this permission for agents, say, will be able to see agents within the user interface, but will not be able to modify them or create another unless granted additional permissions.
Execute	Enables users to run processes associated with applications, components, environments, and resources. Users must also have read permission for an item before actually executing it.

Default Roles

Serena Release Automation ships with several role types mapped to product areas. Every area or type has a set of available permissions. The *application* type, for instance, has the Manage Snapshots permission in addition to the

common permissions (see the [Common Permissions \[page 129\]](#) table. User-defined roles within a type can choose from among the permissions available for that type.

Every product area has one role typically called `Admin` or `Administrator` that has all permissions available for that area. Deleting a default Admin role for one role type does not affect the Admin role for another type.

You can quickly grant a role type's permissions to all users using the Default Permissions tab.

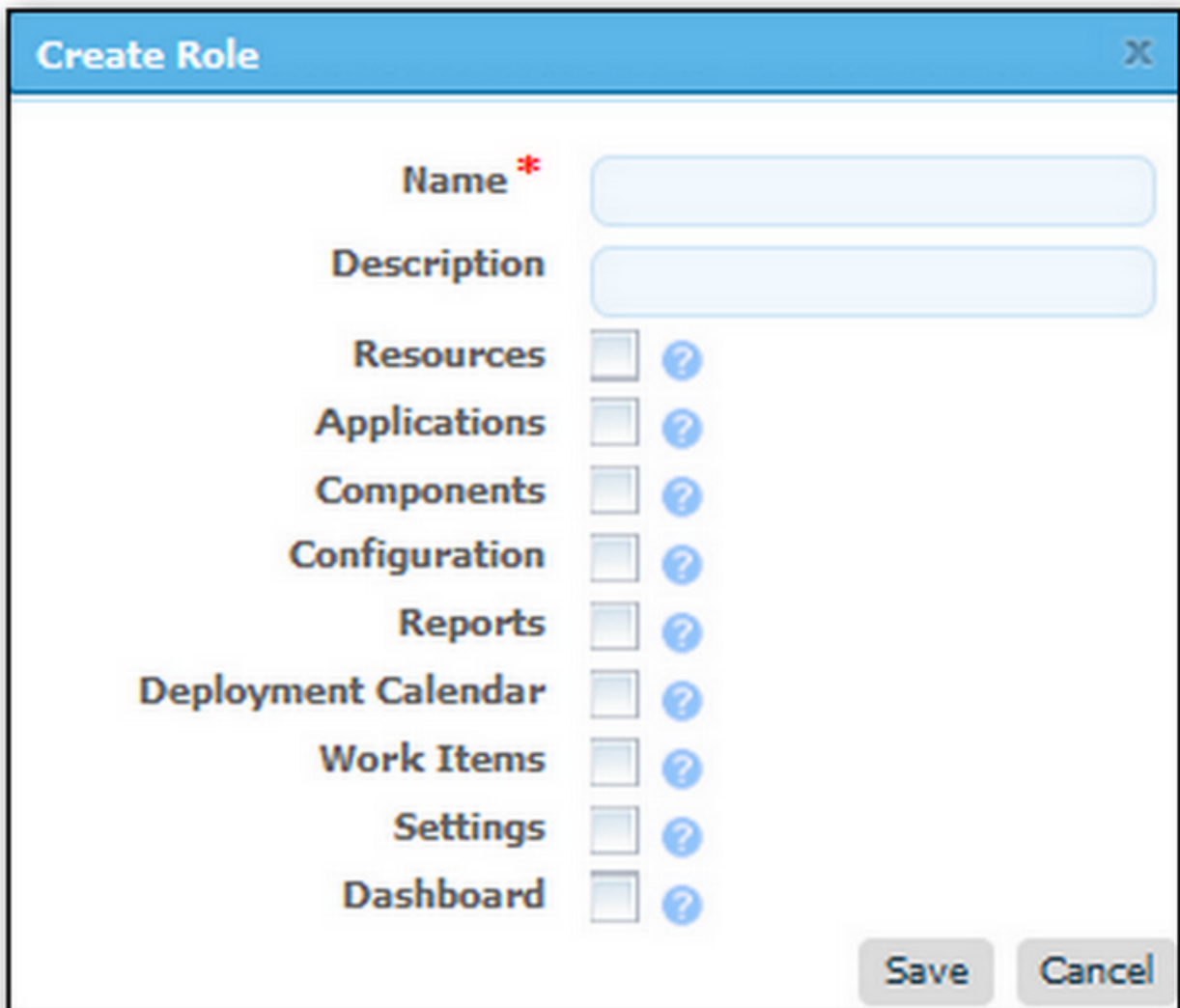
NOTE

Default permissions cannot be granted for system and UI security.

Creating and Editing Roles

1. Display the Role Configuration pane `Settings > Security Role Configuration`.
2. From the list of product areas, select the area where you want to add a role.
3. Display the Create Role dialog `Create Role [button]`.

Application Role Permissions



All permission available for this product display.

4. Select the permissions you want granted to this role.

All roles have the following permissions available. Other permissions—if any—are described in the following sections.

Permissions Available for Every Role table

Permission	Description
Security	Manage security for the effected feature area.
Write	Create, edit, or delete items for this product area.
Read	Access or view items for this product area.

Agent Roles

Agent roles define the functions users can perform with agents and agent pools. Available permissions are read, write, and security, (see [Permissions Available for Every Role \[page 131\]](#)).

To add users to agent roles:

1. Display the Security tab for the target agent `Resources > Agents/Agent Pools > [selected agent/agent pool] > Security`.

All roles defined for agents and agent pools are displayed.

2. Use the Add Role Member action for a specific role, then select the user.

All users are available. As shipped, Serena Release Automation provides an `Admin` role with all configured permissions granted. By default, `Admin` has a single user—`admin`.

Application Roles

Application roles define the functions users can perform with applications. In addition to the standard permissions (see [Permissions Available for Every Role table \[page 131\]](#)), others are:

Application Roles table

Permission	Description
Manage Snapshots	Create and edit snapshots for this application.
Run Component Processes	Run associated component processes outside of the application.

To add users to application roles:

1. Display the Security tab for the target application (see [Applications \[page 95\]](#)) `Applications > [selected application] > Security`.

All defined roles are displayed.

2. Use the Add Role Member action for a specific role, then select the user.

All users are available. As shipped, Serena Release Automation provides an `Admin` role with all configured permissions granted. By default, `Admin` has a single user—`admin`.

Component Template Roles

These roles define the functions users can perform with component templates. Available permissions are read, write, and security (see [Permissions Available for Every Role \[page 131\]](#)).

To add users to component template roles:

1. Display the Security tab for the target template `Components > Templates > [selected template] > Security` (see [Component Templates \[page 86\]](#)).

All defined roles are displayed.

2. Use the Add Role Member action for a specific role, then select the user.

All users are available. As shipped, Serena Release Automation provides an `Admin` role with all configured permissions granted. By default, `Admin` has a single user—`admin`.

Component Roles

These roles define the functions users can perform with components. In addition to the standard permissions, others are available:

Component Roles table

Permission	Description
Manage Versions	Create and delete versions for this component.

To add users to component roles:

1. Display the Security tab for the target component (see [Creating Components \[page 67\]](#)) `Components > [selected component] > Security`.

All defined roles are displayed.

2. Use the Add Role Member action for a specific role, then select the user.

All users are available. As shipped, Serena Release Automation provides an `Admin` role with all configured permissions granted. By default, `Admin` has a single user—`admin`.

Environment Roles

These roles define the functions users can perform with environments (see the [Permissions Available for Every Role table](#)). Available permissions are read, write, execute, and security. [\[page 131\]](#)

To add users to environment roles:

1. Display the Security tab for the target environment `Components > [selected component] > Security`.

All defined roles are displayed.

2. Use the Add Role Member action for a specific role, then select the user.

All users are available. As shipped, Serena Release Automation provides an `Admin` role with all configured permissions granted. By default, `Admin` has a single user—`admin`.

License Roles

These roles define the functions users can perform with licenses. Available permissions are read, write, and security (see the [Permissions Available for Every Role table \[page 131\]](#)).

To add users to license roles:

1. Display the Security tab for licenses (see [Licenses \[page 140\]](#)) `Settings > Licenses > Security`.

All defined roles are displayed.

2. Use the Add Role Member action for a specific role, then select the user.

All users are available. As shipped, Serena Release Automation provides an `Admin` role with all configured permissions granted.

By default, `Admin` has a single user—`admin`.

Resource Roles

These roles define the functions users can perform with resources. Available permissions are read, write, execute, and security, (see [Permissions Available for Every Role table \[page 131\]](#)).

To add users to resource roles:

1. Display the Security tab for the target resource `Resources > [selected resource] > Security`). For resource groups: `Resources > Resource Groups > [Edit Group action] > Security`.

All defined roles are displayed.

2. Use the Add Role Member action for a specific role, then select the user.

All users are available. As shipped, Serena Release Automation provides `Admin` role with all configured permissions granted. By default, `Admin` has a single user—`admin`.

Default Permissions

Default permissions can be set globally for all users for a product area, or for individual user groups within an area. By default, a product areas' permissions are *not* enabled for any user or group (except for the `admin` user which has all permissions for all role types granted). Use the Default Permissions tab to set default permissions, for both the groups you create and those shipped with the product.

Users added to a group inherit the group's default permissions.

Setting Default Permissions

To set default permissions:

1. Display the Default Permissions pane `Settings > Default Permissions`.
2. From the list of product areas, select the area you want to use.

Default Permissions for Agent Area

The screenshot shows the SERENA Administration console. The breadcrumb trail is Home > Settings > Security. The 'Default Permissions' tab is active. On the left, the 'Agent' menu item is selected. The main content area is titled 'Default Permissions for All Users' and contains three checkboxes: Security, Write, and Read. Below this is a section titled 'Default Permissions for Groups' which contains a table with the following data:

Group	Security	Write	Read
Admin Group	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Approve Group	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Configuration Group	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Deploy Group	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

The table also includes a 'Show Filters' link and a pagination control showing '10 per page' and '4 records (total)'.

Selecting an area displays the permissions available for it. User-defined groups are configured independently (see [Authorization Realms \[page 134\]](#)).

3. Check the permissions you want to grant for the selected group.

The following table lists the available permission.

Product Area Privileges table

Role	Read	Write	Security	Execute	Snapshots	Comp.	Proccs.	Versions
Agent	X	X	X					
Agent Pool	X	X	X					
Application	X	X	X	X	X	X		
Component	X	X	X	X				X
Component Template	X	X	X					
Environment	X	X	X	X				
License	X	X	X					
Resource	X	X	X	X				
Resource Group	X	X	X	X				

Authorization Realms

The Authorization Realms pane is used to create user groups and authorization realms. Authorization realms associate users with roles and work with authentication realms to determine which users can access Serena Release Automation. The authorization realms available are:

Internal Storage

Uses internal role management. The default authorization realm—*Internal Security*—is this type.

LDAP

Uses external LDAP role management.

Creating an LDAP Authorization Realm

An LDAP authorization realm uses an external LDAP server for authorization.

To create an LDAP authorization realm:

1. Display the Create Authorization Realm dialog `Settings > Security > Authorization > Create Authorization Realm [button]`.
2. Ensure that `LDAP` is selected in the Type list box, then specify the following:

LDAP Authorization Realm Properties table

Field	Description
User Group Attribute	Name of the attribute that contains role names in the user directory entry. If user groups are defined in LDAP as an attribute of the user, the Group Attribute configuration must be used
Group Search Base	Base directory used to execute group searches, such as <code>ou=employees,dc=mydomain,dc=com</code> .
Group Search Filter	LDAP filter expression used when searching for user entries. The name will be substituted in place of <code>0</code> in the pattern, such as <code>uid={0}</code> . If this is not part of the DN pattern, wrap the value in parenthesis, such as <code>ud=(0)</code> .
Group Name	Directory name used to bind to LDAP for searches, such as <code>cn=Manager,dc=mycompany,dc=com</code> . If not specified, an anonymous connection will be made. Required if the LDAP server cannot be anonymously accessed.
Search Group Subtree	Searches the subtree for the roles if checked.

Creating Authorization Groups

Groups are logical containers that serve as a mechanism to grant permissions to multiple users; members automatically share a group's permissions (see [Default Permissions \[page 133\]](#)). Permissions are granted to groups (or all users), not individual users. Additionally, when a group is assigned a role, its members are automatically assigned the role as well (see [Roles and Permissions \[page 129\]](#)).

To create a group:

1. Display the Create Group dialog `Settings > Security > Authorization > Groups > Create Group [button]`.
2. Provide a name for the group.
The name appears in the Default Permissions pane (see [Default Permissions \[page 133\]](#)).
3. Select an authorization realm.

Groups are only valid for the selected realm.

Serena Release Automation provides several default groups and users, which are listed in the following table. The default groups and users are part of the internal security authorization realm.

Default Groups table

Group	Users
Admin Group	admin
Approve Group	approve
Configuration Group	config
Deploy Group	deploy

Authentication Realms

The Authentication Realms pane is used to create authentication realms and users. Authentication realms determine a user's identity within an authorization realm. Authentication is determined following the hierarchy of realms displayed on the Authentication Realms pane. In the example below, authentication will first be determined in the Internal Security realm followed by the LDAP realm. A user listed in the LDAP realm may have different authorizations from those in the other realms.

Authentication Realms Precedence

The screenshot shows the SERENA user interface. The top navigation bar includes 'Dashboard', 'Components', 'Applications', 'Configuration', 'Processes', 'Resources', 'Calendar', and 'W'. The breadcrumb trail is 'Home > Settings > Security'. The 'Authentication' tab is circled in red. Below it, the 'Internal Security' realm is selected, and its 'operation Tools' (up, down, and delete arrows) are highlighted with a red box. The 'Users' section displays a table of users with columns for User, Name, Email, and Actions. The table lists four users: admin, approve, config, and deploy. Each user has 'Edit', 'Reset Password', and 'Remove' actions. A 'Create User' button is visible at the bottom of the table.

User	Name	Email	Actions
admin	admin		Edit Reset Password Remove
approve	approve		Edit Reset Password Remove
config	config		Edit Reset Password Remove
deploy	deploy		Edit Reset Password Remove

If you have a number of authentication realms, you can reorder them using the operation tools. Each realm can be moved up to a higher priority, moved down, or deleted by using the operation tools.

Creating an Authentication Realm

1. Display the Create New Authentication Realm Settings > Security > Authentication > Create New Realm.

2. Enter a name and description and other basic parameters:

Allowed Login Attempts Number of attempts allowed. A value of 0 means unlimited attempts.

Authorization Realm Requires that the authorization realm was previously created.

Type Selecting `Internal Storage` completes the process.

Creating an LDAP Authentication Realm

If you create an `LDAP` type of authentication realm, you need to provide information about your LDAP installation:

LDAP Authentication Realm Properties table

Field	Description
Context Factory	Context factory class used. This may vary depending upon your Java implementation. The default for Sun Java implementations: <code>com.sun.jndi.ldap.LdapCtxFactory</code> .
LDAP URL	URL to the LDAP server beginning with <code>ldap://</code> or <code>ldaps://</code> . Separate additional servers with spaces.
Use DN Pattern	User directory entry pattern; the name will be substituted in place of <code>0</code> in the pattern, such as <code>cn={0},ou=employees,dc=yourcompany,dc=com</code> .
User Search Base	Base directory used to execute group searches, such as <code>ou=employees,dc=mydomain,dc=com</code> .
User Search Filter	LDAP filter expression used when searching for user entries. The name will be substituted in place of <code>0</code> in the pattern, such as <code>uid={0}</code> . If this is not part of the DN pattern, wrap the value in parenthesis, such as <code>ud=(0)</code> .
Search User Subtree	If the LDAP user names are case sensitive, check the box to treat different-case names as different users.
Search Connection DN	Directory name used to bind to LDAP for searches, such as <code>cn=Manager,dc=mycompany,dc=com</code> . If not specified, an anonymous connection will be made. Required if the LDAP server cannot be anonymously accessed.
Search Connection Password	Password used when connecting to LDAP to perform searches.
Name Attribute	Contains the user's name, as set in LDAP.
Email Attribute	Contains the user's email address, as set in LDAP.

Once configuration is complete, when a new user logs on using their LDAP credentials, they will be listed on the Authentication Realm Users pane. It is best practice not to manage user passwords nor remove users from the list. If an active user is removed from Serena Release Automation, they will still be able to log onto the server as long as their LDAP credentials are valid.

Authentication Realm Users

When adding a new user, the user name and password is what the individual will use when logging into Serena Release Automation. The user name will also be displayed when setting up additional security.

Once the new user has been successfully added to a group, you might need to configure additional permissions. This can happen when the new user is mapped to a group that has limited permissions.

Importing LDAP Users

Unless using LDAP authorization realm, valid LDAP users can log on but will have no permissions. To provide permissions, import them first and define their permissions before they log on. You can import users from existing LDAP systems into Serena Release Automation-managed authentication realms.

To Import LDAP Users

1. Display the Create User dialog `Settings > Security > Authentication Realms > [select LDAP realm] > Import User [button]`.
2. Enter the name of the user.

If you enter a search filter in the `Username` field, the filter must be enclosed in parentheses.

Tokens

Tokens provide authorization for agents and users. Agents use tokens when performing process steps and communicating with the Serena Release Automation server and external services. Users can use tokens with the CLI client (see [Command Line Client \(CLI\) \[page 166\]](#)), and instead of supplying a user name and password in certain situations.

You can create tokens in addition to those shipped with the product.

To create a token:

1. Display the Create New Token dialog `Settings > Security > Tokens > Create New Token [button]`.
2. From the User drop-down list box, select the user who will use the token.
3. Specify the expiration date and time.

Tokens can be used immediately after being created.

User Interface Security

These roles determine which parts of the Serena Release Automation web application users can access. Each tab, such as Reports, on the web application's home page can be restricted. Available permissions are:

Web UI Permissions table

Permission	Description
Resources	Access the Resources tab (see Resources [page 90]).
Applications	Access the Applications tab (see Applications [page 95]).
Components	Access the Components tab (see Components [page 67]).
Configuration	Access the Configuration tab (see Configurations).
Reports	Access the Reports tab (see Reports [page 114]).
Deployment Calendar	Access the Calendar tab (see Scheduling Deployments [page 113]).
Work Items	Access the Work Items tab (see Work Items [page 106]).
Settings	Access the Settings tab [page 139] .
Dashboard	Access the Dashboard tab.

To add users to Web UI roles:

1. >Display the System Security tab `Settings > Security > Security`).
 For resource groups: `Resources > Resource Groups > [Edit Group action] > Security`.
 All defined roles are displayed.
2. Use the Add Role Member action for a specific role, then select the user.

All users are available. As shipped, Serena Release Automation provides the following roles:

Default Web UI Roles table

Role	Description
Deployment Engineer	Access the Reports, Calendar, Work Items, and Dashboard tabs.
Approver	Access the Reports, Work Items, and Dashboard tabs.
Administrator	Access all tabs.
Configuration Engineer	Access all tabs except <i>Calendar</i> and <i>Work Items</i> .

System Security

These roles define the functions users can perform with the Serena Release Automation server (also referred to as system security). Available permissions are:

Server Permissions table

Permission	System Settings Description
Security	Manage security configuration; users without this permission cannot access or change the security functions.
Manage Plug-ins	Grants users the ability to install new plug-ins (see Installing Plug-ins [page 146]).
Create Subresources	Ability to create subresources (see Resources [page 90]).
Create\Manage Resource Roles	Create and delete resource roles (see Resource Roles [page 92]).
Create Components	Create components (see Creating Components [page 67]).
Create Applications	Create applications (see Creating Applications [page 97]).
Create Component Templates	Create component templates (see Creating a Component Template [page 86]).
Manage Licenses	Add and remove licenses (see Licenses [page 140]).

To add users to system security roles:

1. Display the System Security tab `Settings > Security > Security`.

All defined roles display.

2. Use the **Add Role Member** action for a specific role, then select the user.

All users are available.

As shipped, Serena Release Automation provides `Configuration Manager` and `System Administrator` roles; the latter has all configured permissions granted. By default, `System Administrator` role has a single group—`Admin Group` (with user `admin`), and the `Configuration Manager` role also has a single group—`Configuration Group` (with user `config`).

System Settings

You can use the System Settings area of the Serena Release Automation web application to access, create, and/or modify features such as:

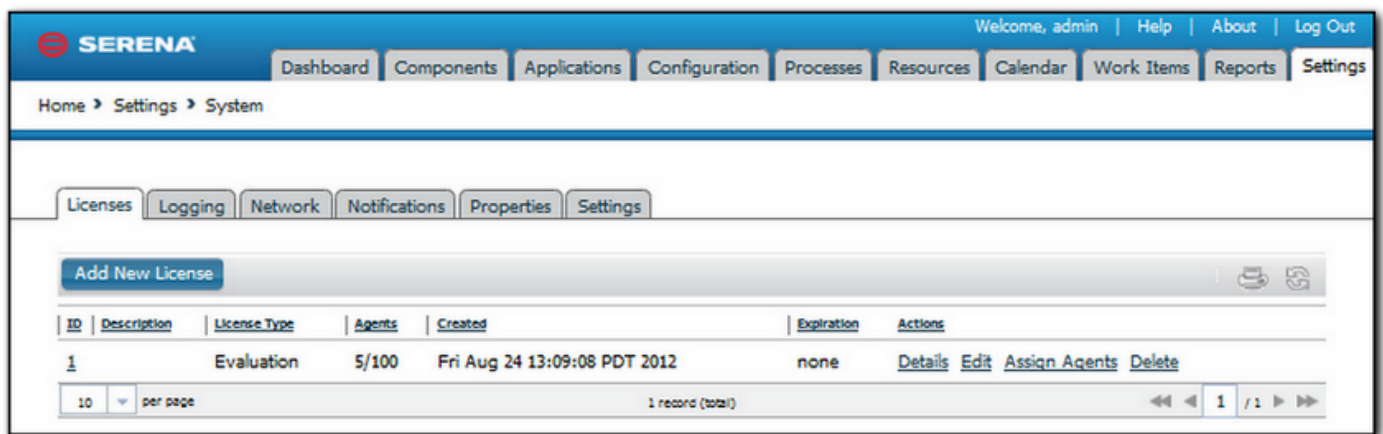
- licensing
- logs

- network relays
- email notifications
- plug-in step post-processing scripts
- plug-ins installation

Licenses

The Licenses pane is where you manage user licenses (adding or deleting licenses, and assigning agents to them). Display the Licenses pane by clicking the Licenses link on the Settings window `Home > Settings > License`. You can also access the pane through the Resources tab `Resources > Agents > License`.

Licenses Pane



Adding a License

To add a license:

1. Display the Add New License dialog by clicking the Add New License button.
2. Paste the license text supplied by Serena into the License field.
3. (Optional) Add a description.
4. Click Save when you are done.

To see information about a license, display the License Details pop-up by clicking the Details link.

Adding Agents to a License

Agents can be assigned to licenses automatically or manually. This section explains how to add agents manually. To automatically add agents, ensure that the Automatic License Management check box on the System Settings pane is checked (see [the System Settings \[page 139\]](#)).

To add an agent to a license manually:

1. Display the Assign Agents to License pop-up by clicking the Assign Agents link for the intended license.

2. >Click the Agents field.
A selection-type pop-up displays.
3. Make a selection from the list of agents that are not already assigned to the license.
4. Select the agent or agents you want to add to the license.
5. Optional. You can filter the listed agents by entering search text into the text field.
6. After select agents, click OK to close the selection pop-up.
7. If you want to restart the selection process, click Reset.
8. When you are finished, click Save.

Modifying or Deleting a License

To modify or update an existing license:

1. Display the Edit License dialog by clicking the Edit link for the license you want to change.
2. Edit the information shown in the License field.
3. Click Save.

To delete an existing license, click the Delete link for the selected license.

Logging

You can download the Serena Release Automation server log from within the web application.

To download the log file:

1. Display the Output Log dialog by clicking the Output Log link on the Settings pane.
2. Click the Download Log button to save the file.
3. Optional. You can download the file directly from the Settings pane by clicking the Download link on the Settings pane.

The Serena Release Automation server log file is normally found in the following location: `Serena Release Automation_root\var\log\deployserver.out`.

Network Relay

A *network relay* is used in conjunction with an agent relay. The network relay reverses the direction of communication through a firewall between the Serena Release Automation server and agent relay. A network relay is only used when you want the server to connect to the relay instead of the reverse (which is default). To create a network relay an agent relay must be created. (See [Installing Agent Relays \[page 44\]](#) to create an agent relay)

To creating a network relay:

1. Display the network pane `Home > Settings > System > Create New Network Relay`
2. Enter the name of the network relay.

3. Identify the Host and Port.
4. Indicate the Network Relay will be Active by checking the box.

Notifications

Serena Release Automation can send email notifications whenever user-defined trigger events occur. Notifications can be sent when a deployment finishes or an approval is required, for example. Notification recipients are defined with the security system's (see [Serena Release Automation Security \[page 128\]](#)) LDAP integration. If you have not already done so, set up LDAP prior to configuring notifications. Serena Release Automation relies on LDAP and an associated e-mail server to send notifications.

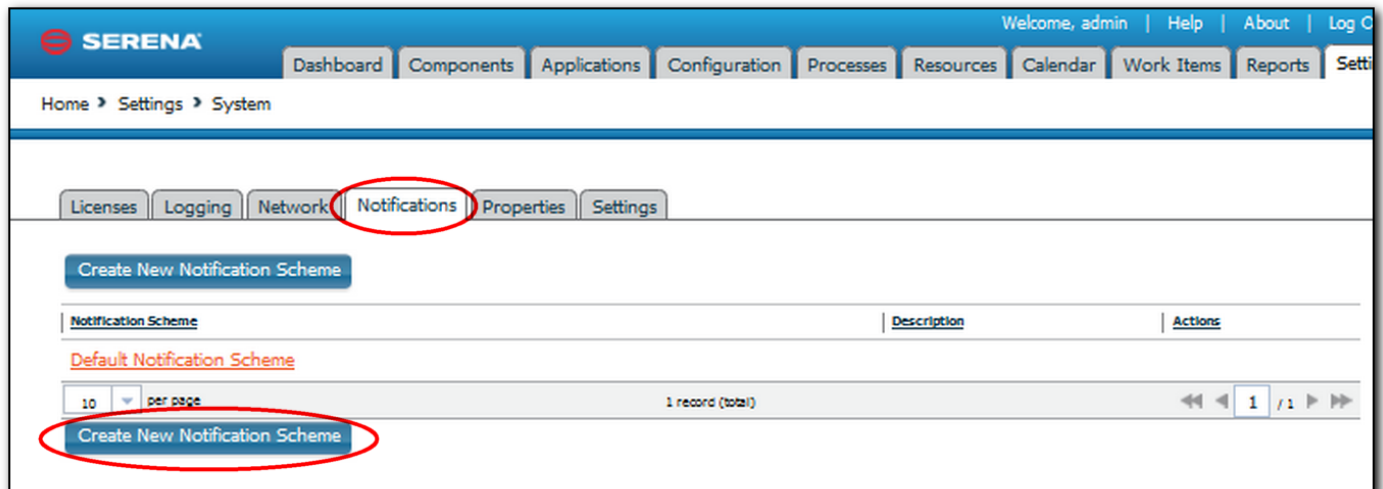
NOTE

Serena Release Automation requires an external SMTP mail server to send notifications. For information about configuring a mail server, see [System Settings](#).

When setting up notifications, you select both the triggering events and the role, which is inherited from the security system, to determine which users will receive notification. For example, it is common for an administrator or environment owner to be notified when a work item (as part of the approval process) has been generated. The default notification scheme, which sends notifications to the application and admin default roles (see [Serena Release Automation Security \[page 128\]](#)), can be edited or you can create your own scheme.

To set up your own notifications, display the Notifications pane [Settings > Notifications](#).

Notification Schemes



The screenshot shows the Serena Release Automation web interface. The top navigation bar includes the Serena logo and user information (Welcome, admin | Help | About | Log Out). Below the navigation bar, there are tabs for Dashboard, Components, Applications, Configuration, Processes, Resources, Calendar, Work Items, Reports, and Settings. The current page is 'Settings > System'. Under the 'System' settings, there are tabs for Licenses, Logging, Network, Notifications (highlighted with a red circle), Properties, and Settings. Below these tabs, there is a 'Create New Notification Scheme' button. A table lists the 'Default Notification Scheme' with columns for 'Notification Scheme', 'Description', and 'Actions'. Below the table, there is a pagination control showing '10 per page' and '1 record (total)'. Another 'Create New Notification Scheme' button is highlighted with a red circle at the bottom of the table area.

Configure the new Scheme. Here, you will be setting up the who/when for notifications. Once configured, you can come back add additional Entries to the Scheme or edit an existing one.

Notification Type

The process type is determined mainly by the type of recipient. For example, a deployment engineer would be interested in being notified about a failed deployment.

Add Notification Entry

Type * **Process Success** ?

Target * **Process Success** ?

Role *

Template Name *

Process Failure ?

Process Started ?

Approval Completed ?

Approval Failed ?

Save Cancel

Notification Target

When setting the target, the application option will only send out notifications when the event selected above corresponds to an Application. For example, the "Process Success" event, when paired with the "Application" Target would trigger a notification when a Process (an application deployment) is successful. Similarly, the same event type, when used with the "Environment" target would instigate a notification when a successful deployment has been run in an Environment, such as, SIT, PROD).

Add Notification Entry

Type * **Process Success** ?

Target * **Application** ?

Role *

Template Name * **DeploymentFailure** ?

Application ?

Environment ?

Save Cancel

Notification Role

The Role corresponds to those set in the Security System. Any individual assigned the Role you select will receive an e-mail.

Add Notification Entry

Type * **Process Success** ?

Target * **Application** ?

Role * **Admin** ?

Template Name * **DeploymentFailure** ?

Admin ?

Save Cancel

Template Name

The available templates are provided by default and should meet all your needs, including formatting the email being sent. Which template you use is based on why you want to set up a notification and the recipients of the notification. However, if the default templates do not suit your needs, you can create your own.

Template	Description
Application deployment failure/success	Sends notifications about a specific application to the specified users, based on the role setting.
Task readied/created/completed	Used to report back on the state of manual tasks.
Deployment readied	A specialized email template for letting people know a deployment has been prepared.
Approval created/failed	These templates are used to notify the status of an approval.

Once you have the entry done, add others using the same process. If you want to use the new notification scheme with existing applications, modify the application settings.

Creating Notification Templates

Notification Templates are XML files located in the server's `conf/server/notification-template` file folder. If the default notification templates do not suit your needs, you can create new ones.

To create a new notification template:

1. Start a new XML file.
2. Enter Script. (Notification templates only supports Velocity Reports)
3. Save file in the server's `conf/server/notification-template` file folder.
4. Restart the server.

Post-Processing Scripts

Serena Release Automation component processes perform post-processing whenever a plug-in step finishes execution. Typically, post-processing scripts ensure that expected results occurred. You can use your own JavaScript script instead by instructing Serena Release Automation to use your script when you define the step, see [Process Editor \[page 75\]](#).

When a step finishes, the agent performing the step will run your script (the script must be written in JavaScript). When the agent runs the script, it first loads the server log file and finds the exit code property of the target step using regular expressions defined in the script. It then applies any actions defined in the script before processing the next step.

To create a script:

1. Display the **Edit Script** dialog `Settings > Post Processing Scripts`.
2. Enter a name for the script into the `Name` field. The name must match the name you specified when you defined the process step, (see [Process Editor \[page 75\]](#)).
3. Enter or paste the script into the `Script Body` field. See the roll-over help next to the field for information about the properties and variables available for user-defined scripts.

The Serena Release Automation server log file is normally found in the following location: `Serena Release Automation_root\var\log\deployserver.out`.

Preview Version Cleanup

To preview the component versions that will be archived the next time an archive file is created, click the `Preview Version Cleanup` link. Using the link displays the Version Cleanup Preview dialog, which lists the to-be-archived component versions.

Output Log

You can download the Serena Release Automation server log from within the web application.

To download the log file:

1. Display the Output Log dialog by clicking the `Output Log` link on the Settings pane.
2. Click Download Log to save the file.
3. Optional. You can download the file directly from the Settings pane by clicking the `Download` link on the Settings pane.

The Serena Release Automation server log file default location is: `Serena Release Automation_root\var\log\deployserver.out`.

Locks

A *lock* is a routinely used to ensure that processes do not interfere with one another. Normally, once a lock is no longer needed it is released. Sometimes a lock will not get released and its associated process will be unable to complete. The lock management feature enables you to quickly identify and resolve abnormal lock conditions.

Managing Locks

A running process with a lock, like all active processes, appears on the Dashboard tab with a status of `Running`. If a locked process takes longer to complete than expected, you can cancel the process from the Dashboard, or investigate it fully with the `Settings` tab.

1. Display the Lock pane by clicking the `Locks` link on the Settings tab `Home > Settings > Locks`.

The Lock pane displays the following information:

Lock Fields table

Field	Description
Name	The name identifies the lock. The displayed name is a concatenation of the component or application name (depending on type) + process name + resource name.
Type	Indicates whether the process creating the lock is a component- or application-type. Locks can only be applied to component or application processes.
Component/ Application	Displays the name of the component or application containing the lock. Clicking an item displays (depending on the type) the Component pane, or Application pane, where you can investigate the lock.
Resource/ Environment	Displays the name of the resource or environment containing the lock. Clicking an item displays (depending on the type) the Resource pane, or Environment pane.
Process	Displays the name of the process containing the lock. Clicking an item displays the process in the process editor.
Actions	Lists the available actions.

2. Resolve the lock by selecting an action:

Lock Actions table

Action	Description
View	Displays the process log for the process containing the lock. You can use the <code>Actions</code> field on the Request displayed pane to see the name of the process step causing the lock.
Release	Releases the lock which enables the associated process to continue processing.

If the Serena Release Automation server and or agents go down while a locked process is running, Serena Release Automation will automatically restore any interrupted processes along with any locks they might contain once service is restored.

Installing Plug-ins

Plug-ins can be installed at any time. You can download a zip file that contains all the plug-ins from the <http://www.serenasupport.com> page.

To download the plug-ins:

1. Go to <http://support.serena.com/Case/CaseHome.aspx> and log in using your customer account.
2. Browse to the My Downloads tab.
3. From the Please Select Product drop-down, select `Serena Release Automation`.
4. Download the plug-ins zip file to the platform on which you want to deploy it.

NOTE

You *do not* need to decompress the .zip file.

To install the downloaded plug-ins:

1. From the Automation Plug-ins pane, display the Load Plug-in dialog `Settings > Plugins > Load Plugin [button]`.
2. Enter the path to the compressed plug-in (.zip) file and click Submit.

After the plug-ins load process completes successfully, the plug-ins are listed on the Automation Plug-ins pane. Once installed, plug-in functionality is available immediately.

Configuration

The Serena Release Automation Configuration tool enables you to directly manage application, component, and environment configuration data.

Configuration data is manipulated at the application, component, and environment levels:

Component

A component refers to any file that you want to include in the build process; components are associated with the configuration data required to deploy them.

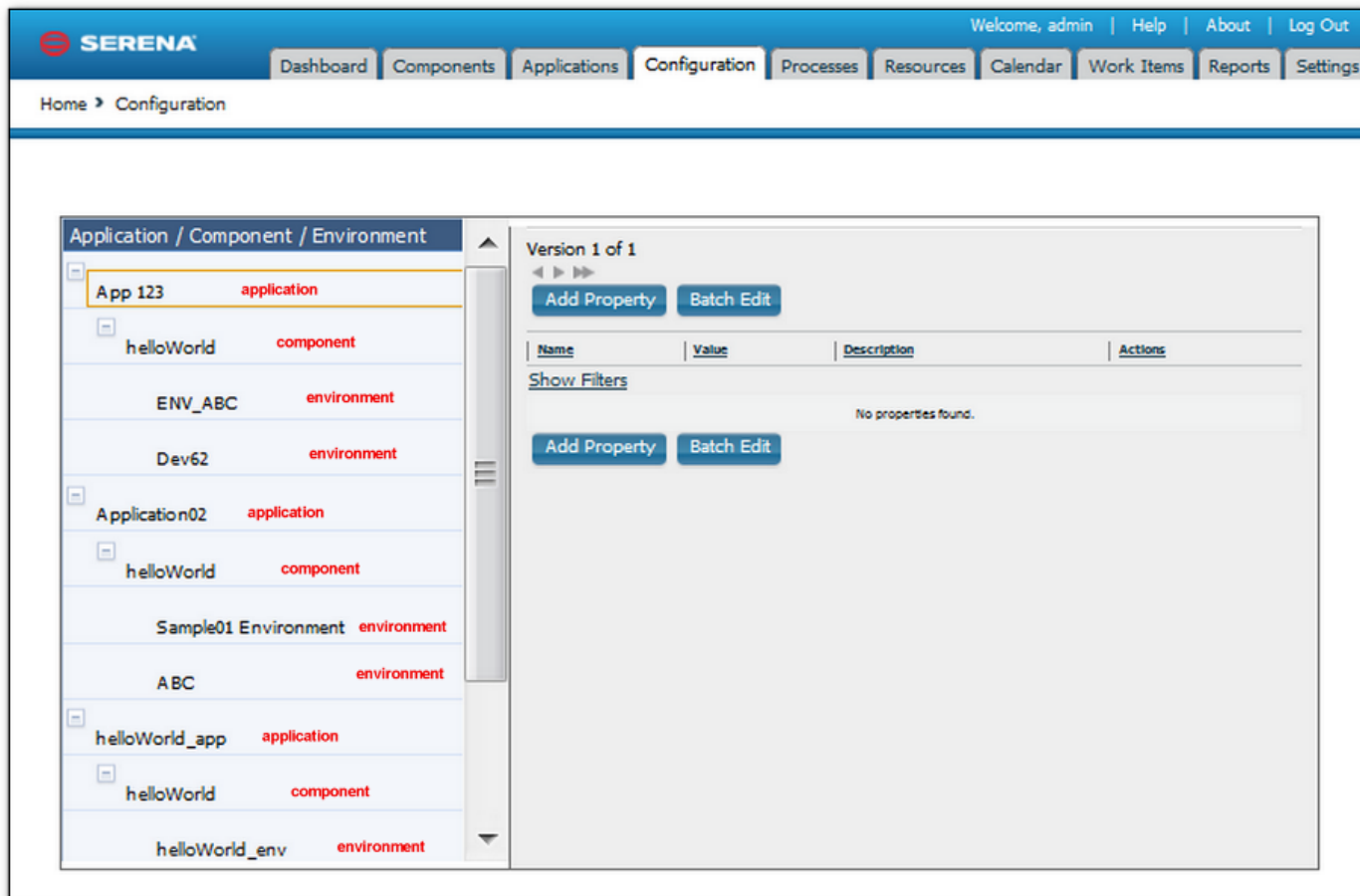
Application

Applications represent a group of components deployed together by component version and environment. Applications also map the hosts and machines (called resources) components require within every environment.

Environment

An environment is a collection of resources that host a Serena Release Automation application.

Configuration tab

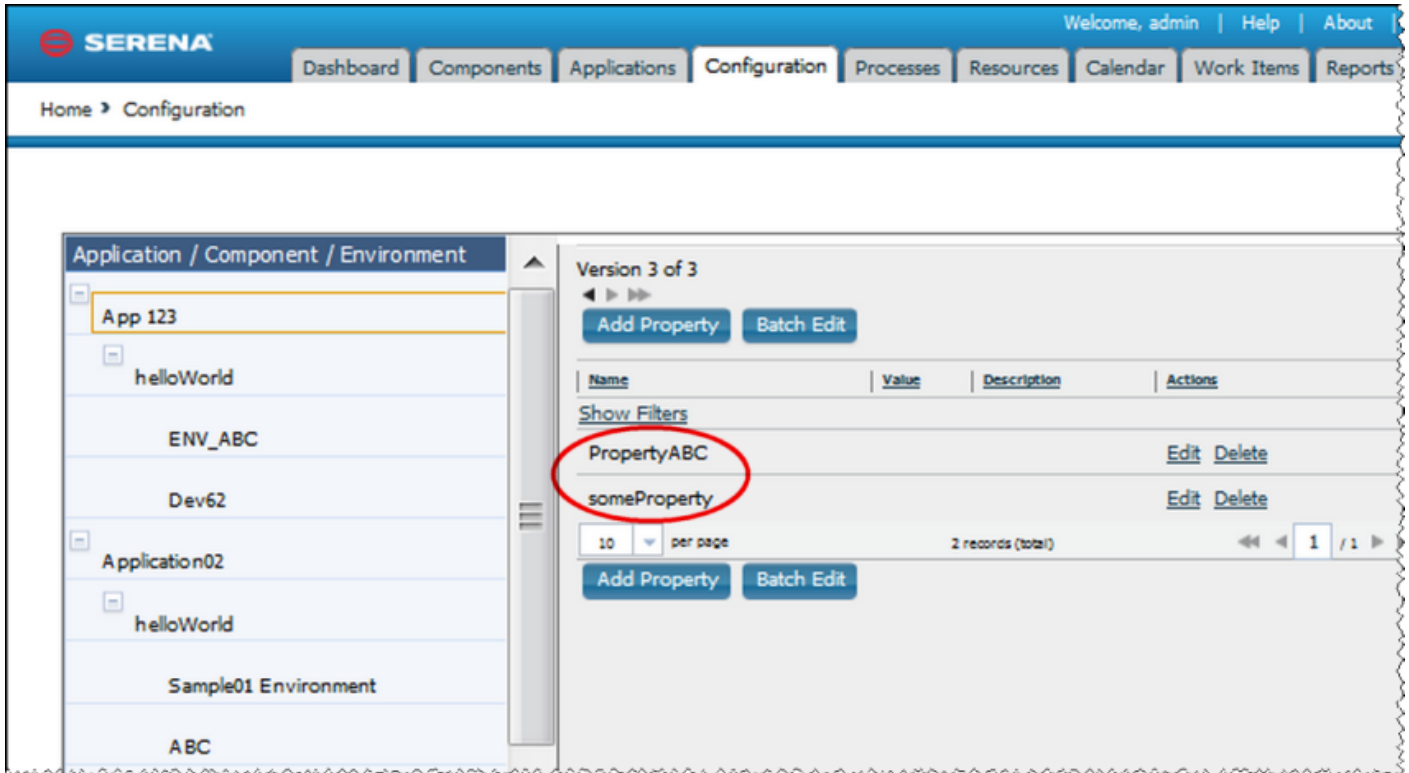


Access the Configuration Tool by clicking on the Configuration tab.

Application Configuration

You attach properties to an application by using the Configuration Tool's *Application: Add Property* button. Typical application-level properties include items that are the same in all environments, such as base-install paths.

Application Properties panel



The screenshot shows the SERENA Configuration Tool interface. The top navigation bar includes 'SERENA' and 'Welcome, admin | Help | About'. Below the navigation bar are tabs for 'Dashboard', 'Components', 'Applications', 'Configuration', 'Processes', 'Resources', 'Calendar', 'Work Items', and 'Reports'. The breadcrumb trail shows 'Home > Configuration'. On the left, the 'Application / Component / Environment' list is expanded to show 'App 123' selected, with sub-items 'helloWorld', 'ENV_ABC', 'Dev62', 'Application02', 'helloWorld', 'Sample01 Environment', and 'ABC'. On the right, the 'Version 3 of 3' configuration panel is displayed. It features 'Add Property' and 'Batch Edit' buttons. Below these is a table with columns 'Name', 'Value', 'Description', and 'Actions'. The table contains two rows: 'PropertyABC' and 'someProperty', both with 'Edit' and 'Delete' links. A red circle highlights the 'someProperty' row. Below the table, there is a '10 per page' dropdown, '2 records (total)', and pagination controls showing '1 / 1'. Another 'Add Property' and 'Batch Edit' button is located at the bottom of the panel.

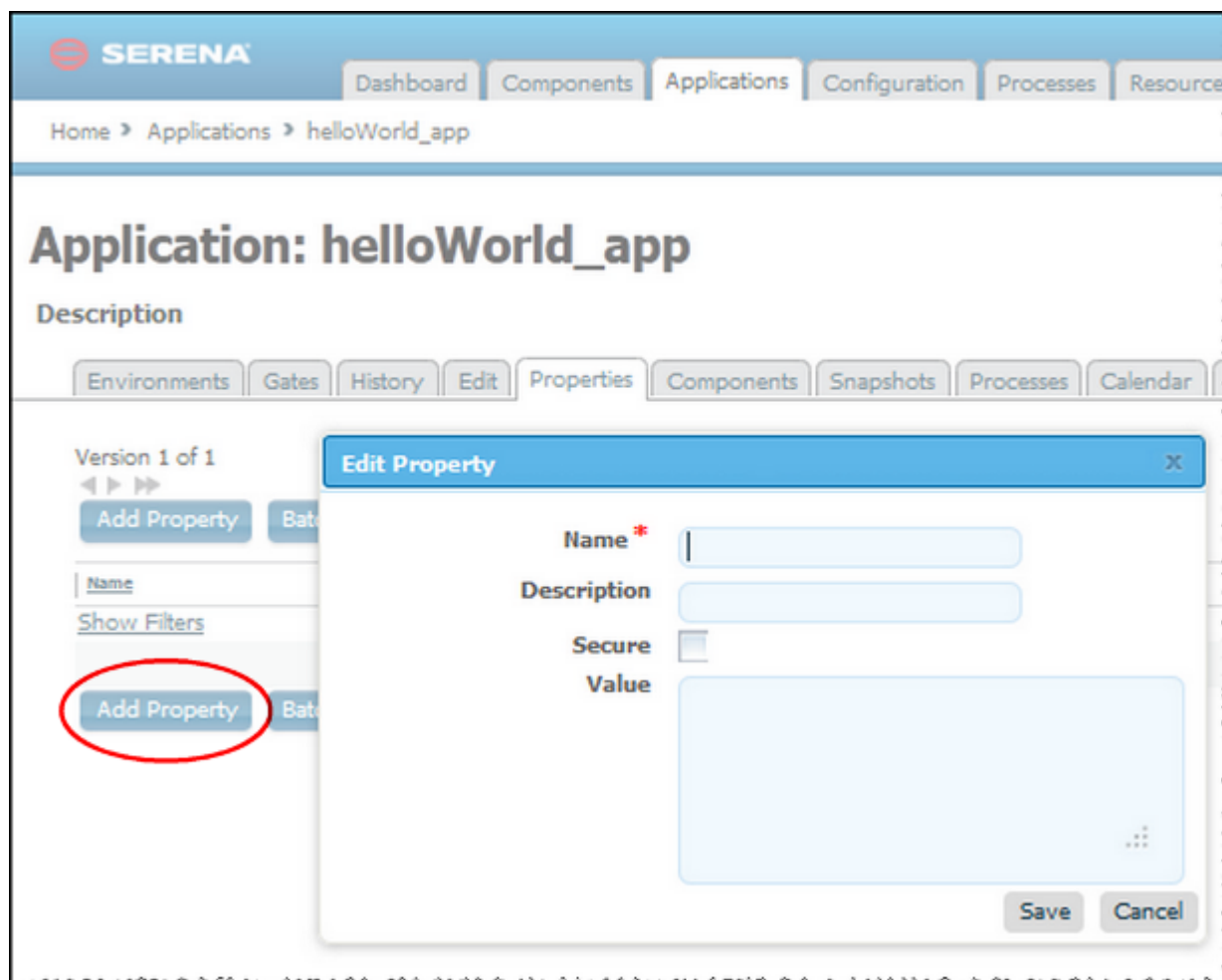
Access the Configuration Tool Application panel by clicking on an application in the Application/Component/Environment list box.

Adding Application Configuration Properties

To add a property to the selected application:

1. Use the Add Property button.

The Edit Property pop-up displays.

Edit Property pop-up

2. Enter the property's name in the Name field.

While component fields can be of any size, configuration properties are restricted to 4,000 characters.

3. Enter a description of the property in the Description field.
4. Specify whether the property is secure by using the Secure check box.

Secure properties are stored encrypted and displayed obscured in Serena Release Automation's user interface.

5. Enter a value for the property in the Value field.
6. To save the property, click Save, or to discard your work click Cancel.

Modifying and Deleting Application Configuration Properties

Modifying Application Configuration Properties

To modify a previously created property, use the `Edit` link in the Action column to display the Edit Property pop-up.

Deleting Application Configuration Properties

To delete a property, use the `Delete` link in the Action column.

Component Configuration

The Serena Release Automation Configuration tab enables you to configure applications and their components from a single location. Configuration data is manipulated at the application, component, and environment levels:

component

A component refers to any file that you want to include in the build process; components are associated with the configuration data required to deploy them.

application

Applications represent a group of components deployed together by component version and environment. Applications also map the hosts and machines (called resources) components require within every environment.

environment

An environment is a collection of resources that host a Serena Release Automation application.

Access the Configuration Tool by clicking on the Configuration tab.

Environment Configuration

The Serena Release Automation Configuration tab enables you to configure applications and their components from a single location. Configuration data is manipulated at the application, component, and environment levels:

component

A component refers to any file that you want to include in the build process; components are associated with the configuration data required to deploy them.

application

Applications represent a group of components deployed together by component version and environment. Applications also map the hosts and machines (called resources) components require within every environment.

environment

An environment is a collection of resources that host a Serena Release Automation application.

Access the Configuration Tool by clicking on the Configuration tab.

Inventory

The *inventory* shows what applications and components have been deployed, including the current versions that are running on the resource within an environment. The inventory provides complete visibility into the different versions of your applications which can be tracked back to the original artifacts imported into Serena Release Automation.

There are different views of the current inventory, depending on where in Serena Release Automation you are. Inventory information is available on the individual Components, for every application environment, as well as for each resource (agent).

Resources Inventory

If you want to see what Components are sitting on the SIT Environment, go to Resources and select the agent that is running in the Environment. From here, selecting either the Component or its Version will take you to the Component's page if you need more information.

Resource inventory

The screenshot shows the SERENA interface for the 'Deploy PROD' agent. The page title is 'Resource: Deploy PROD'. Below the title, there are tabs for 'Main', 'Inventory', 'Edit', 'Roles & Properties', 'Changes', and 'Security'. The 'Inventory' tab is selected. A message states 'No resources are using this agent.' Below this, there is a 'Create New Resource' button. A table lists resources with the following columns: Process, Component, Version, Application, Environment, Scheduled For, Status, and Actions. Two rows are shown, both with a 'Success' status. A red circle highlights the 'Component' and 'Version' columns for both rows.

Process	Component	Version	Application	Environment	Scheduled For	Status	Actions
hello_worldInstall	helloWorld	3	helloWorld_app	helloWorld_env	10/15/12 1:56 PM	Success	View Request
hello_worldInstall	helloWorld	remote-agent05	helloWorld_app	helloWorld_env	10/15/12 12:55 PM	Success	View Request

10 per page 2 records (total) 1 / 1

Component Inventory

Unlike the Resource Inventory, the *component inventory* tells you what version of the Component is running on a Resource. For example, if the Component is currently deployed to multiple machines, they would all be displayed. From here, you can navigate to the Resource.

Component Inventory

The screenshot shows the SERENA web interface. At the top, there is a navigation bar with the SERENA logo and user information: 'Welcome, admin | Help | About | Log Out'. Below this is a secondary navigation bar with tabs: 'Dashboard', 'Components', 'Applications', 'Configuration', 'Processes', 'Resources', 'Calendar', 'Work Items', 'Reports', and 'Settings'. The breadcrumb trail reads 'Home > Components > helloWorld'.

The main heading is 'Component: helloWorld'. Below it, the 'Used By' section lists: 'App_123, Application02, QuarterEnd, helloWorld_app, helloWorld_application (Inactive)'. A row of tabs includes 'History', 'Edit', 'Inventory' (circled in red), 'Calendar', 'Properties', 'Templates', 'Versions', 'Processes', 'Tasks', 'Changes', and 'Security'.

There are two tables. The first table lists applications:

Application	Description
App_123	
Application02	
QuarterEnd	
helloWorld_app	
helloWorld_application	

Below the first table is a pagination control: '10 per page', '3 records (total)', and navigation arrows.

The second table lists resources:

Resource	Version	Date	Status	Actions
Deploy_PROD (circled in red)	3	10/15/12 1:56 PM	Active	
Agent1534 (circled in red)	4	10/10/12 1:07 PM	Active	

Below the second table is a pagination control: '10 per page', '2 records (total)', and navigation arrows.

Environment Inventory

For any given Application Environment, the *environment inventory* tells you both what version of any given Component is running on a particular Resource. If multiple Versions are running on different Resources, they will all be listed.

Reference

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.

Fields Available for All Source Types table

Field	Description
Name	Identifies the component; appears in many UI features. Required.
Description	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.
Template	<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 <code>None</code>.</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 Component Templates [page 86].</p> <p>Note</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>
Source Config Type	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 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.
Import Versions Automatically	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.
Copy to CodeStation	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. Serena recommends that the box be checked.
Default Version Type	Required. 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> .

Field	Description
Inherit Cleanup Settings	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)

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.

File System (Basic) Source Fields table

Field	Description
Base Path	Defines how the property is presented to users in the Serena Release Automation editor. This element has several attributes.
Always Use Name Pattern	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.
Version Name Pattern	Defines how the property is presented to users in the Serena Release Automation editor. This element has several attributes.
Next Version Number	Defines how the property is presented to users in the Serena Release Automation editor. This element has several attributes.
Save File Execute Bits	Defines how the property is presented to users in the Serena Release Automation 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.

File System (Versioned) Source Fields table

Field	Description
Base Path	Path to directory containing artifacts. The content of each subdirectory within the base directory is considered a distinct component version. 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 Serena Release Automation 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 [Creating Components \[page 67\]](#).

See [Basic Fields \[page 153\]](#) for information about the standard fields which apply to each source type.

Serena Dimensions Fields table

Field	Description
Username	Dimensions CM user name. For information about user impersonation, see User Impersonation [page 26] .
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

Serena Release Automation 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 Serena Release Automation editor, you can create fully-automated deployment processes. In addition to basic plug-ins, others integrate many third-party tools into Serena Release Automation, such as application servers and software configuration management products. For example, the Tomcat and WebSphere plug-ins 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 [Creating Plug-ins \[page 158\]](#)).

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 [Component Processes \[page 74\]](#); for information about creating your own post-processing scripts, see [Post-Processing Scripts \[page 144\]](#).

Example Process

Process: Deploy Application

Description

Deploys my Web Application

Version

2 of 2



Design

Edit


Properties

Changes

Tools



Available Plugin Steps

-  Finish
- + Utility Steps
- + Application Server
- + Automation
- + Build Systems
- + Cloud
- + Database
- + Infrastructure
- + Installers
- + ITSM Process
- + Middleware
- + Quality
- + Repositories
- + Resource Tools



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 [User Impersonation \[page 26\]](#)).
- 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 [Agent Installation \[page 43\]](#).

Standard Plug-ins

Serena Release Automation 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 Integration

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.

Serena Package Manager

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

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 Serena Release Automation 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. Although optional, Serena 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 Serena Release Automation to make it available to users.

To upload a plug-in

1. Create a ZIP archive that contains the XML files (plugin.xml and upgrade.xml) along with any scripts required by the plug-in.
2. Import the ZIP file with the Automation Plug-ins 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 Serena Release Automation 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.serena.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>
```

<header> Element

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

<header> Child Elements	Description
	<p>This element's three attributes identify the plug-in:</p> <ul style="list-style-type: none"> • <i>version</i> API version (the version number used for upgrading plug-ins is defined in the <code>info.xml</code> file). • <i>id</i>
<code><identifier></code>	<p>Identifies the plug-in.</p> <ul style="list-style-type: none"> • <i>name</i> The plug-in name appears on Serena Release Automation's web application Automation Plugins pane. <p>All values must be enclosed within single-quotes.</p>
<code><description></code>	<p>Describes the plug-in; appears on Serena Release Automation's web application Automation Plugins pane.</p>
<code><tag></code>	<p>Defines where the plug-in is listed within the Serena Release Automation 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.</p>

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: <step-type> Element

Plug-in steps are defined with the `step-type` element; each `step-type` represents a single step in the Serena Release Automation 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 Serena Release Automation's web application.

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


Step Properties: <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.

<property> Element table

<property> Child Elements	Description
<property- ui>	<p>Defines how the property is presented to users in the Serana Release Automation editor. This element has several attributes:</p> <ul style="list-style-type: none"> • <code>label</code> Identifies the property in the editor's Edit Properties dialog box. • <code>description</code> Text displayed to users in the associated roll-over help box. • <code>default-value</code> Property value displayed when the Edit Properties dialog box is displayed; used if unchanged. • <code>type</code> Identifies the type of widget displayed to users. Possible values are: <ul style="list-style-type: none"> ◦ <code>textBox</code> Enables users to enter an arbitrary amount of text, limited to 4064 characters. ◦ <code>textAreaBox</code> Enables users to enter an arbitrary amount of text (larger input area than <code>textBox</code>), limited to limited to 4064 characters. ◦ <code>secureBox</code> Used for passwords. Similar to <code>textBox</code> except values are redacted. ◦ <code>checkBox</code> Displays a check box. If checked, a value of <code>true</code> will be used; otherwise the property is not set. ◦ <code>selectBox</code> Requires a list of one or more values which will be displayed in a drop-down list box. Configuring a value is described below.
<value>	<p>Used to specify values for a <code>selectBox</code>. 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.</p>

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>
```

<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:

<arg> Element Attributes table

Attribute	Description
<code><value></code>	Specifies a parameter passed to the tool. Format is tool-specific; must be enclosed by single-quotes.
<code><path></code>	Path to files or classes required by the tool. Must be enclosed by single-quotes.
<code><file></code>	Specifies the path to any files or classes required by the tool. Format is tool-specific; must be enclosed by single-quotes.

Because `<arg>` elements are processed in the order they are defined, ensured 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 [Serena Release Automation Properties \[page 164\]](#)) 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 Serena Release Automation editor, see [Post-Processing Scripts \[page 144\]](#). Although not required, Serena recommends 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 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:

1. Increment the number of the `version` attribute of the `<identifier>` element in `plugin.xml`.
2. Create a `<migrate>` element in `upgrade.xml` with a `to-version` attribute containing the new number.
3. 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.

Serena Release Automation Properties

A 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.

Every item from the table below will use this format: `${p:version.name}`

Serena Release Automation Properties table

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 Serena Release Automation assigns to distinguish the component from others. The component id is created when a component is created in Serena Release Automation.
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 Serena Release Automation.
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 Serena Release Automation 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 agents name can be entered by editing the agent's <code>conf/agent/installed.properties</code> file and restarting the agent.
stepname/propertyname	Used in output from step
<i>property_name</i>	Component or application process property; defined on the process's Properties tab. Given value by whoever runs the process.
component/ <i>property_name</i>	Component custom property; set on the component's Properties tab.
environment/ <i>property_name</i>	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.
resource/ <i>property_name</i>	Resource properties. This can include the built-in agent properties as well as any custom properties. Each of these have their own tab on the resource.
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 page in the Settings area.

All of the following are comma-separated series of name=value, including each property on the given object. This is useful for token replacement.

Name/Value Pairs table

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.

Property	Description
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.

Command Line Client (CLI)

CLI is a command-line interface that provides access to the Serena Release Automation server. It can be used to find or set properties, and perform numerous functions, described below.

To install the tool, download the `serena_ra-4.0.1.4384.zip` from the Serena Release Automation release page on Supportal (<http://support.Serena.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:

Argument	Description
-authtoken,	Optional. Can be set via the environment variable <code>DS_AUTH_TOKEN</code> . An authentication token generated by the server. Either an authtoken or a username and password is required.
-authtoken	
-password,	Optional. Can be set via the environment variable <code>DS_PASSWORD</code> . A password to authenticate with the server. Either an authtoken or a username and password is required.
-password	
-username,	Optional. Can be set via the environment variable <code>DS_USERNAME</code> . A username to authenticate with the server. Either an authtoken or a username and password is required.
-username	
-webur,	Required. Can be set via the environment variable <code>DS_WEB_URL</code> . The base URL of the Serena Release Automation server— <code>http://ds.domain.com:8585</code> .
-webur	

The global flags are:

Flag	Description
-t,	Show the JSON template for the command instead of running the command. If a file argument is <code>-getTemplateprovided</code> , the template will be output to that file.
-getTemplateprovided	
-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 -webur http://localhost:8080 -username admin -password admin getResources
```

Commands

NOTE

CLI commands do not support new lines. Sample entries 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

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

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.

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

deleteGroup

Delete a group

Format

```
udclient [global-args...] [global-flags...]  
deleteGroup [args...]
```

Options

-group, --group
Required. Name of the group

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

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 for an application request.

Format

```
udclient [global-args...] [global-flags...]  
getApplicationProcessRequestStatus [args...]
```

Options

`-request, --request`
Required. ID of the application process request to view

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

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

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

getEnvironmentsInApplication

Get all environments in an application

Format

```
udclient [global-args...] [global-flags...]  
getEnvironmentsInApplication [args...]
```

Options

`-application, --application`
Required. Name of the application to get environments for

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

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.

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

getResources

Get a JSONArray representation of all resources

Format

```
udclient [global-args...] [global-flags...]  
getResources [args...]
```

Options

No options for this command.

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

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

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.

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

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
```

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 [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
```

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

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.

setComponentEnvironmentProperty

Set property on component/environment mapping

Format

```
udclient [global-args...] [global-flags...]  
setComponentEnvironmentProperty [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  
  
-environment, --environment  
    Required. Name or id of the environment to look up  
  
-application, --application  
    Optional. 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
```

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
```

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
```