

AnthillPro Documentation

AnthillPro Documentation

I. Concepts	1
1. Projects	3
2. Workflows	4
3. Jobs	6
4. Build Life	7
5. Dependencies and Artifacts	8
6. Environments	9
7. Life-Cycle Models	10
II. Getting Started	11
8. Installation	12
9. Setting Up the Server	51
10. Create a New Project	56
11. Setting Up Continuous Integration	67
12. Basic Notifications	70
13. Setting Up a Deployment	72
14. Defining a Basic Dependency	80
15. Integrating Other Tools	81
III. Upgrade	83
16. Server Upgrade	84
17. Migrate Server Database	85
18. Agent Upgrade	88
19. Distributed Servers Upgrade	89
20. Agent Relay Upgrade	90
21. What's New in AnthillPro 5.0	91
22. Uninstalling AnthillPro	94
Uninstall the Server	94
Uninstall the Agent	94
IV. Schedules	97
23. Create Interval Schedule	98
24. Create Cron Schedule	99
V. Reporting	100
25. Running a Report	101
26. Creating a Template Report	102
27. Creating a Velocity Report	103
28. Writing a Report Template	104
29. RSS and Reporting	107
VI. Dependency Management	108
30. Configure Dependencies	110
31. Using Codestation Projects	120
32. Configure Dependencies Tutorial	122
VII. Advanced Authoring	130
33. Managing Projects	131
34. Authoring Workflows	133
35. Administrating Jobs	152
36. Reusing Workflow Definitions and Jobs	154
37. Using Life-Cycle Models	158
38. Notifications	166
39. Properties	176
40. Build Life Links	201
VIII. Advanced Usage	203
41. Build Life Tools	204
42. Workflow Tools	212
43. Job Tools	218
44. User Views	219
45. System Tray Monitor	222
IX. Developer Tools	223
46. Dev-kit	224

47. IDE Plugins	226
48. Preflight Builds	227
49. Codestation for Developers	228
X. Server Management	232
50. Server Settings	233
51. Backups of AnthillPro	238
52. Optimizing Server Performance	244
53. Create Server Proxy	248
54. Clone AnthillPro Instance	249
55. Moving the AnthillPro Database	251
56. Distributed Servers	252
57. Using Externally Signed Certificate with Tomcat	257
XI. Agent Management	258
58. Agent Configuration	262
59. Environment Management	264
60. Configure and Edit Agent Filters	269
61. Agent Settings	271
62. Dynamically Set Agent's Environment Variables	273
XII. Tool Integrations	274
63. SCM Tools	275
64. Build Tools	348
65. Testing Tools	362
66. Coverage Tools	401
67. Source-code Analysis Tools	415
68. Issue Tracking Tools	439
69. Virtualization Tools	520
70. Using AnthillPro Plugins	524
XIII. Security	526
71. Setting Up Security	527
72. Securing Artifact Sets	541
73. Set Up and Manage Guest Users	543
74. Perform Security Audits	544

Part I. Concepts

AnthillPro is a complete, extensible platform for build & deployment management that enables you to create consistency throughout the entire application lifecycle -- from development to release -- based on your current processes:

- **Build management.** AnthillPro allows for distributed and multi-platform building. Every time AnthillPro builds a project, it publishes the artifacts -- including docs and binaries -- to AnthillPro's embedded artifact and dependency management system. This allows AnthillPro to provide extensive information regarding your builds, especially for Continuous Integration: Most of the SCM integrations allow for commit triggers, changelog parsing, etc.

When properly set up, many -- if not all -- pieces of your builds will be repeatable, standardized, and traceable because your current processes are automated as part of the configuration process. See [Setting Up a Build](#) to get started.

- **Deployment management.** AnthillPro's deployment management features allow you to use the same deployment scripts across all your environments, so you can automate your deployment processes while enforcing standards throughout your organization.

AnthillPro's binary promotion model and artifact repository ensure that you are using the exact same binaries from environment to environment. And to account for environmental differences, AnthillPro allows you to set job-iteration, environment-specific properties, etc. See [Setting Up a Deployment](#) for more on how AnthillPro deals with deployment management.

- **Dependency and artifact management.** Complex projects can depend on dozens of subprojects, and even simple projects might use half a dozen libraries or more. To deal with these complexities, AnthillPro has a built-in dependency and artifact management system called Codestation.

Codestation enables you to track and visualize dependencies across libraries and subprojects based on version numbers or release statuses that you control. Because Codestation is part of AnthillPro, there is no need for a third-party management system -- dependency management is performed through the AnthillPro UI. (If you don't use AnthillPro for dependency management, you can still use it for build and deployment management, as well as process automation.)

Codestation is also used to manage the artifacts generated by every build. Through the UI, you can configure which files are available as a dependency to other projects, which files should be included in a deployment, as well as the files you want sent to the testing lab. See [Dependency Management](#) for more.

- **Tool integration and data aggregation.** AnthillPro integrates with leading tools used throughout the application lifecycle. Most integrations go beyond a simple command-line integration to include data aggregation (e.g., parsing of the output, knowledge of the tool options, etc.).

A complete list of the current integrations is available on the UrbanCode website here [<http://www.urbancode.com/html/products/build/integrations/default.html>].

For example, when AnthillPro gets the changelog from an SCM, that information is stored in the AnthillPro data warehouse and then made available through the UI. For integrations with issue trackers, AnthillPro can add a comment to an existing issue, or even create an issue in the other tool. And for testing integrations, AnthillPro collects test results and can then provide you with metrics for both a single build and for many builds over time -- or even fail a build based on test results. See [Tool Integrations](#) for more on any of the individual integrations.

- **Process automation.** As you start building projects and configure AnthillPro with your SCM, testing, issue tracking tools and more, it becomes clear how all of the parts fit together to automate your processes: AnthillPro en-
-

forces automation throughout the entire application life-cycle, which in turn makes managing your projects easier. For more advanced users, AnthillPro offers host of advanced authoring features that enable you to add additional automation to your AnthillPro processes: Including the creation of automated operations or processes, full access to the API (see **Tools > Anthill Development Tools**) and the ability to write your own Plugin.

Chapter 1. Projects

In AnthillPro, there are three project types: Life-Cycle Based, used to run builds, deployments, etc.; Codestation Projects, used to manage dependencies of third-party tool kits and software libraries; and Operational (Non-Life-Cycle Based) Projects, used for administrative, operational, and system maintenance. Following is a brief outline of the major components of each project type:

- **Life-Cycle Based Projects** provide a wealth of information and visibility into the build and release cycle. The Life-Cycle Based project is used when configuring builds, and always creates a Build Life (in its simplest terms, the Build Life can be thought of as a build number) every time a build is run. Life-Cycle Based projects correspond to components of your application, and build independently of each other. For example, a typical web application might have UI, database, and logic tiers. In AnthillPro, these can be mapped as three projects that build separately. Configured in this way, you don't have to build the entire application to make a simple UI change, etc. The majority of the projects you set up in AnthillPro will be Life-Cycle Based. A Life-Cycle Based Project is defined by its:
 - *Life-Cycle Model*. Provides a template for managing the dependencies, artifacts, deployments, etc., associated with every build of the project. Because a Life-Cycle Model is reusable, it allows you to apply the same standards to any similar projects. See Life-Cycle Models for more.
 - *Environments*. A project will usually participate in many environments that correspond to the different stages the project goes through. For example, most organizations will have something similar to DEV, QA, and PROD environments that correspond to their different teams, etc. Essentially, a project's environment groups is the collection of environments it uses. If a project will never build on or deploy to certain environments, an environment group can be used to help segregate the project. See Environments for more.
 - *Build, or originating, workflow*. Defines the process for running jobs. When manually starting a build or scheduling work, a workflow is being executed. Each originating workflow will contain at least one job that runs your build script, etc. The originating workflow is also where you let AnthillPro know what specific source code is being built by the project, as well as configuring any dependencies. Any secondary workflows (to perform deployments, testing, etc.) configured for a project will inherit much of the information from the build workflow. The best way to learn about originating workflows is to set up a build process.
- **Codestation Projects** model products not built by your organization as an AnthillPro project. This allows the use of third-party tools as dependencies, testing them as new releases come out to track their approval status within the Application.

In addition to making stored artifacts available to the build process, Codestation Projects also make the configured dependency graph, and the artifacts themselves, available to developers. Through either an IDE Plugin, a command-line utility, or Ant tasks, a Codestation resolve operation on the developer's machine fetches the appropriate artifacts. See Using Codestation Projects and Codestation (Developers).

- **Operational (Non-Life-Cycle Based) Projects** provide a simplified interface that allows AnthillPro users to execute ancillary tasks not easily run during a build, deployment, etc., workflow. Because of their flexibility, Operational Projects can be used to automate most tasks associated with the traditional build and release cycles. For example, an Operational Project can be used to verify the integrity of deployed artifacts; start and recycle services on restricted machines; and automate other process that do not fit neatly into a traditional build. Steps can be run in parallel, on different agents, execute arbitrary commands, integrate with third-party tools, execute scripts, or perform any task or series of tasks that a normal project can perform. Similar to Life-Cycle Based projects, Operational Projects are defined by the environments they participate in and their workflows. See Using Operational Projects.

Chapter 2. Workflows

A workflow is a series of jobs that should be run, and determines how, when and where the jobs should be run. Workflows assemble jobs into processes, and are the unit of automation. Workflows, also manage the order and parallelization of jobs.

As a rule, when manually starting a build or scheduling work, a workflow is being executed. Workflows assemble many of the other configurable items into something usable:

- A **job** is a series of steps detailing how to get something done. It may specify the steps in a build, deployment, or other business need. The job configuration also dictates what types and number of machines the job will be run on (a deployment job may target every machine in a cluster).
- A **trigger** is an automated mechanism for kicking off a workflow.
- **Source Configuration** identifies exactly which source code should be retrieved from a repository. How these are configured will vary between source code management tools.

To start, AnthillPro has two basic types: originating and non-origination. **Originating workflows** are used to start Build Lives, utilizing Life-Cycle Models, source configuration, and dependency information. **Non-origination workflows** specify secondary processes that can be run on an existing Build Life (e.g., a deployment). Underneath, the difference is that originating workflows have corresponding BuildProfile objects.

Components of a Workflow:

- **Workflow Definition.** Defines which job should be run, specifies the order of jobs, as well as the elements to be run in parallel.
- **Environment.** For builds, this will almost always be the Build-Farm. For workflows that handle deployments, these might be deployment targets like QA Servers or production.
 - An originating workflow may only select one environment, while a non-originating can select any number of target environments.
- **Notification Scheme.** Determines who to notify of build status, what conditions to notify them on, and what mechanism to notify them with.
- **Lockable Resource.** Specifies a resource that might be used by multiple workflows, gives it a name, and forces arbitrary workflows to run one at a time or in small groups.
- **Properties.** Identifies various properties that must be specified when the workflow is executed. See Workflow Property.
- **Triggers.** Allows workflow to be kicked off automatically.
- **Source Configuration (Originating Only).** Specifies the source that should be used for the workflow. Often this means selecting a branch of your project to run a workflow against. By creating a workflow for each branch, the stamping strategy used for each can be unique, while activities and jobs only need to be configured once.
- **Stamp Mapping (Originating Only).** For every stamp style in the project's stamp style group, one must specify what stamping strategy will be used. A typical stamp style group might specify the styles of development and production. In the workflow, you map the concept of a style used for development builds to a version number and strategy for incrementing that number.
- **Dependencies (Originating Only).** Specifies the specific artifacts, including version, from other projects that

should be retrieved in order to support the workflow. Also specifies the artifacts from other projects and the directory in the checked-out source that those artifacts should be placed in.

- **Artifacts (Originating Only).** For any artifact associated with this project, the files produced by a workflow published as the artifact.

Workflows can either be created for individual projects or be used by multiple projects. To use the same Workflow for multiple projects, a Workflow Library (Workflow Definition) is created. Once configured, Workflow Definitions are added to existing projects as part of the project-workflow creation process (on the project-workflow Definition tab). Workflow Definitions do not necessarily have to be placed in Workflow Library folder. Any Workflows (on the Administration page) not in a folder or as part of a project subdirectory are Workflow Definitions.

Chapter 3. Jobs

The job is composed of a series of distinct actions the server must perform (called "steps" in AnthillPro) to successfully run a build, deployment, or any other AnthillPro process. For any job you create, the steps (actions) are executed by the server one at a time, in a specific order.

Job configuration comprises a major component of a project. Each job step may be laid out explicitly or generated implicitly by a special job definition. Typical jobs are "build", "deploy", or "run functional tests". AnthillPro provides a Job Wizard to assist in the creation of build jobs. The Wizard takes you through the steps to configure the builder and the publisher required by the job, and is the best way to ensure your build job will be configured successfully.

To learn about job configuration, see [Create a New Job](#) in the [Setting Up a Build](#) section.

Chapter 4. Build Life

The Build Life represents all the transformations a build has gone through, and the processes such as deployments and testing that the artifacts have undergone.

Whether you are a developer, tester, manager or release engineer, AnthillPro provides comprehensive view (presented in a single location) of your builds from beginning to end: When a build is created, a Build Life number is created. When automated tests are run against the build, the results are recorded under the Build Life. When QA tests a build, their findings are included as part of the Build Life. Finally, when a release is performed, it is the Build Life that provides an exact trace to every change that went into the build, all the testing that was performed on the build, and the different environments the build progressed through.

In effect, the Build Life is a map of (1.) what occurred during the build; (2.) the processes that were performed on the generated artifacts; and (3.) where the build artifacts ended up:

1. **The build.** Often called the pure build, this step takes the source as input and transforms it into artifacts. In AnthillPro, it is the pure build (or simply build) which generates the Build Life -- represented on the dashboard as the Build Life number. The build process is configured as an originating workflow in AnthillPro.
2. **Secondary processes.** Once a build is successful, that is where the Build Life begins to come into play. The generated artifacts are stored in AnthillPro and made available for additional processes -- most commonly testing. For example, many organizations have a suite of tests that run every night to check the stability of the application. When this is the case, a secondary workflow is configured to execute the processes. The testing process inherits a majority of its configuration properties from the build (originating) workflow. Once the test suite is complete, the data is collected and presented on the AnthillPro dashboard, under the same Build Life number generated by the build.
3. **Deployments and releases.** Generally, deployment and release processes are treated much the same way as testing processes. They are usually configured as secondary workflows. Once a deployment or release has been run, the results, including the environments the artifacts were sent to, will be made available on the dashboard's Build Life page.

Much of the information presented as part of the Build Life is determined by processes you configure in AnthillPro: You configure the source and the location of the build, map out the different stages of the build's life-cycle -- including all the testing processes that the build will go through -- and configure dependencies. All of these moving parts are brought together as a Build Life.

Chapter 5. Dependencies and Artifacts

Dependencies between projects are very common. Often a project will produce a library used by another project. When few project dependencies exist, manually adding the products of one project into another is straightforward. However, when dependencies start to get complicated, you need an automated solution.

Dependency relationships are often used to configure build scheduling. Typically, there are three main approaches: use of a scheduling method independent of the dependency graph; a pulling model; and a pushing model:

- **Independent Scheduling.** Related projects do not need to know about each other for independent scheduling. A dependent project starts each build by gathering the most recent artifacts from its dependencies.
- **Push Scheduling.** Also called the bottom-up model. When an originating workflow (that other projects depend on) builds, the workflows that depend on it automatically build. Because changes to the dependency result in changes to the dependent, rapid feedback as to whether the dependent was broken by the changes is available.
- **Pull Scheduling.** The dependent is not automatically built whenever the dependencies are built. Rather, it is built according to its own schedules. When it builds, however, it starts the process by checking to see if there have been any changes to the dependencies and building them first if required. This is helpful to ensure that the latest check-ins are tested together.

In AnthillPro, dependencies between projects are described at the workflow level. Workflows in dependencies specify the location (in the working directory) of the artifacts used by dependent projects. A workflow may be both a dependency and dependent, but cycles are not allowed. When a workflow uses dependency artifacts, AnthillPro maintains a record of which build life the artifacts came from. This provides instant traceability from the dependent project back to its own source files, as well as those of each of its dependencies. See [Configuring Dependencies](#).

AnthillPro provides an internal artifact store to track generated artifacts, tie them to build lives, and provide them to the dependent workflows. This store also makes it easy for a workflow to specify that it needs an artifact from a dependency that comes from a specific build life. For instance, this could be a requirement that the shared library has passed testing. Or an artifact could be locked down to a specific version. This approach is common as a project nears release because it helps prevent the team from making any deleterious changes. See [Using Codestation Projects](#).

- **Codestation.** The name for AnthillPro's artifact repository. In addition to making stored artifacts available to the build process, Codestation also makes the configured dependency graph, and the artifacts themselves, available to developers. Through either an IDE Plugin, a command-line utility, or Ant tasks, a Codestation resolve operation on the developer's machine fetches the appropriate artifacts.
- **Third-party Artifacts.** Codestation provides a utility for uploading approved versions of third-party libraries. It provides these libraries to builds (similar to the way in-house dependencies are managed).

Chapter 6. Environments

An environment is a partition grid of agents that is specific for different stages of a project Life-Cycle (QA, PROD, etc.). Each environment may also be configured to a specific technology (Java, .NET, etc.).

Workflows within AnthillPro execute on a specified environment. This allows the definition of a single deployment workflow that can then be used to deploy the application to separate QA and production environments, etc. The same deployment workflow executed on the QA environment deploys the application to the QA environment. When executed on the production environment, it deploys the application to the production environment.

- **Agents** can participate in multiple environments. This allows the use of an agent that communicates with a network deployment manager capable of deploying applications to multiple environments.

Environments are organized as part of an Environment Group: Each individual environment participates in at least one group, or container. An environment group determines the set of environments that project workflows may be executed on. Each environment group must contain at least one environment (typically models a development, QA, production, or other environment).

Using Environment Groups: Consider an organization that develops software using two different technologies. The first is J2EE. Our hypothetical organization has a development, a QA, and a production J2EE environment, with each containing application servers configured to the corporate standard. The second technology used is C++, with development, QA, and production environments for all C++ projects. The C++ environments use different physical servers than the corresponding J2EE environments. In this scenario, J2EE projects and C++ projects are configured into two environment groups: one called 'J2EE Env Group', and the other called 'C++ Env Group'. The 'J2EE Env Group' contains the J2EE development, QA, and production environment groups, and the 'C++ Env Group' contains the C++ environment groups. With these environment groups, the organization derided any confusion about what set of servers a J2EE or C++ project should be deployed to.

AnthillPro ships with one implied environment: the Build-Farm which can't be deleted. It is the default environment containing all agents used for pure-build processes. Because there are no restrictions on how the Build-Farm is used, it may be used for deployments as well. Additionally, AnthillPro has no restriction to prevent you from using other environment groups for pure build processes.

Chapter 7. Life-Cycle Models

Life-Cycle Model allow you to create a reusable template that maps your organizational structure, giving you control over how a build is identified, the different stages a build must go through on its way to the end user, how artifacts are handled, and which clean-up policies are enforced.

A Life-Cycle Model consists of:

- **Cleanup Policy** specifies when to delete information about old Build Lives and other tasks associated with the project.
- **Stamp Style Group** creates common names for types of stamps.
- **Artifact Set** is a grouping of related build products, such as the scripts used to populate a database, artwork for a video game, a CD ISO, or code library. Because most projects generate similar types of artifacts, the artifact set group holds the names of artifact sets for a type of project.
- **Status Group** is a set of common names for statuses. 'Failed' and 'Successful' are default status names, but common other names might be 'Deployed to Test', 'Deployed to Production', 'Retired', 'Passed Function Testing' or 'Approved'.

A typical life-cycle (or pipeline) would be DEV > QA > PROD: a build starts out in development, is deployed to quality assurance for testing, and then finally sent to production for release. You would configure the Life-Cycle Model to apply a new status when a build is sent to QA, and one when the build is sent to PROD. Likewise, a stamp style (essentially a build identifier) can be applied to each build that corresponds to the status. This enables you to know exactly which build is in which environment because the status and stamp are recorded on the Dashboard. See Using Life-Cycle Models for more.

Part II. Getting Started

This section provides a general introduction to using AnthillPro's build and deployment management capabilities.

Before you begin, it is recommended you read the Concepts section to get a general understanding of the terminology used. For example, it will be helpful to know what is meant by "Lifecycle Model," "Environment (and Environment Group)," "Workflow," and "Job."

The step-by-step instructions -- which, in general, should be followed in the order of presentation -- take a hands-on approach to using AnthillPro:

1. **Install AnthillPro.** Takes you through the different installation scenarios. Since AnthillPro is a Java application, it can run on most any operating system. The system requires you to install the central server, including a backing database, as well as at least one agent.
2. **Set up the server.** Walks through basic server configuration you will most likely need to perform before creating your first project. In general, you will need to activate the license, set up a user and configure security, assign agents to an environment, and then configure at least one repository (and possibly a trigger).
3. **Create a new project.** Begins the process for getting a build going in AnthillPro.
4. **Set up a build.** Covers what you need to know to set up a basic build in AnthillPro. With a build process in place, AnthillPro is configured to provide all you will need for Build Management.
5. **Set up Continuous Integration.** Introduces you to implementing Continuous Integration with AnthillPro. Building upon the "Setting Up a Build Process" section, you can see how to add unit testing to the build, as well as set up a basic notification scheme.
6. **Configure basic notifications.** Shows you the different options for notifying developers (and others) on the state of the build. You can configure the instant message and/or email integrations.
7. **Set up a deployment (secondary or non-originating workflow).** Takes you through the steps necessary to deploy the build artifacts. This section introduces the concept of Artifact Sets, which are used by the AnthillPro Deployment Management system. Once you have a deployment process in place, AnthillPro automatically links every deployment to a build (also called a Build Life).
8. **Add dependencies with another project.** Gives you the basics of how the AnthillPro dependency management system works, including instruction on setting up a simple dependency relationship. Most projects will depend on the artifacts from another project in order to build; however, it is possible to use AnthillPro without the dependency management feature.
9. **Integrating other tools.** Provides a general overview of what tools AnthillPro integrates with, and how the integrations are configured through the UI.

Once you are familiar with AnthillPro basics, the Advanced Authoring and Advanced Usage sections give you a deeper look at extending AnthillPro.

Chapter 8. Installation

To get AnthillPro running, you will need to install Java (on both the server and agent machines), the database, the server, and then the agent.

To evaluate AnthillPro, fill out the request an evaluation copy of AnthillPro [<https://support.urbancode.com>] form. Once your request is approved, you will be provided with login access to Urbancode's support team.

Most users have found the following procedures -- executed in the order presented -- helpful:

1. Review the Install Recommendations section.
2. **Determine minimum system requirements.** Before you get started, please review the System Requirements section. AnthillPro should run on most modern machines; however, if you are installing a production instance the System Requirements section outlines what current users have had success with.
3. **Select a database type.** AnthillPro requires a backing database -- so if you are not using Derby (included in the download for evaluation purposes) you will need to decide which database to use. It is recommended to review the Database Installation section before continuing.
4. **Install Java on the server and agent machines.** Both the server and agent require **Java JRE 5 or greater** be installed on the machine, as well as the JAVA_HOME environment variable set. The JAVA_HOME environment variable must point to the Java directory that you want AnthillPro to use.

You may also use the appropriate version of the JDK. (It is recommended to use the Sun JDK. There have been some issues with the third-party JDK's from IBM and others.) See also 32- and 64-bit JVM Support.

5. **Install the database.** AnthillPro ships with the Derby database, which should be fine for evaluation purposes but is not suitable for most enterprise applications. Currently, AnthillPro supports the following databases to store server and project information:

Apache Derby (included in download)	MySQL with InnoDB storage (4.1.22 and later)
DB2 (requires 9.7 or later)	Oracle, including RAC
Microsoft SQL Server	PostgreSQL

To install the database (except for Derby, which does not require a separate installation) you will need to:

- a. **Download the appropriate JDBC driver file for your database.** These are typically downloaded from the database vendor. If using the Derby database, you can skip this item.
- b. **Create an empty database for AnthillPro to use with a dedicated user.** If using the Derby database, you can skip this item. Follow the instructions given for your database type:

Install Using DB2 (requires 9.7 or later)	Install Using Oracle
Install Using Microsoft SQL Server	Install Using PostgreSQ

6. Next, Download AnthillPro if not already done so.
7. Finally, install the server and agent. See either Windows Installation or Linux/Unix Installation.

Install Recommendations

Since the AnthillPro agent performs most of the heavy lifting (e.g., performing builds, running tests, etc.), agent installation is critical to maximizing performance. Following are a few installation recommendations to reduce the chances of performance-related issues:

- **Install the server as a user account.** Typically, the server should be installed as a dedicated system account when possible. However, AnthillPro will run very well as a Root (or Local System on Windows) and running that way is often the easiest way to start as you never get permissions errors.
- **Install the agent as dedicated system account.** Ideally, the account used should be one created just for AnthillPro. Because AnthillPro agents are remote command execution engines, it is advisable to limit what they can do on each machine by creating a user just for the automation engine and grant it appropriate privileges. For example, if you plan on using the agent to run tests, it will need permissions to the testing tool executable, etc. If you install the agent as Root (or Local System on Windows), ensure that the processes you perform don't wipe out the file system, etc.
- **Do not install an agent on the AnthillPro server machine.** Because the agent is resource intensive, installing one on the server machine will degrade server performance when a build, etc., is running on that agent. The agent is responsible for doing the "heavy lifting." When a build is run, the agent checks out the code, compiles it, performs any other processes you have configured, and then moves the build artifacts.
- **Install only one agent on a given machine where possible.** As with installing an agent on the server box, having multiple agents on the same machine will have negative effects on each other ... resulting in slow builds, etc. This situation often arises when an organization installs multiple agents for different environments on the machine instead of installing one agent and assigning it to multiple environments. However, there may be instances where installing multiple agents is necessary. When this is the case, you will see varying performance when more than one agent is busy.

System Requirements

AnthillPro is OS agnostic, and will run on both Windows and UNIX-like systems. While the minimum requirements should suffice for initial evaluation purposes, it will be necessary to run the AnthillPro server on a server-class machine for most production deployments.

Server

Minimum requirements. The AnthillPro server will run on any modern business machine:

- **Processor.** Single core, 1.5 GHz +.
- **Disk Space.** 300 MB.
- **Memory.** 2 GB, with 256 MB available to AnthillPro.

Recommended system requirements. For larger deployments (e.g., you will be running thousands of processes a day), most users have found the following guidelines helpful:

- **Small to medium sized server-class machines.** It is recommended to have two machines: The primary machine and a cold standby for fail-over.

The back-end database should be hosted on a separate machine, according to vendor guidelines. See also Data-

base Requirements.

- **Processor.** 2 CPU. 2+ core each.
- **RAM.** 8 GB.
- **Storage.** Your individual requirements depend on your usage, retention policies, and types of applications. In general, the more artifacts you keep in AnthillPro's artifact repository (Codestation), the greater the storage needs (e.g., you keep the generated artifacts from every CI build). Additionally, the server logs must also be taken into consideration -- over time log storage can add up.

For further assistance in determining storage requirements, please contact support [<http://support.urbancode.com/>].

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

Agent

Agents are designed to be minimally intrusive, and should only require 64-256 MB of memory and 100 MB on disk. Additional requirements are determined by the processes the agent will run.

For small evaluations, it is possible to install the agent on the same physical machine as the server; however, *this should be avoided for large deployments of AnthillPro*.

For further assistance in determining agent requirements (e.g., a heavily used build machine), please contact support [<http://support.urbancode.com/>].

32- and 64-bit JVM Support

The AnthillPro server must run against the 32-bit JDK for the Windows 2003 64-bit server. However, the 64-bit JDK can be used for agents that run builds. (It is recommended to use the Sun JDK. There have been some issues with the third-party JDK's from IBM and others.) Because AnthillPro does not require a multi-gigabyte heap, there is little advantage to using a 64-bit JVM for any operating system. However, for those on a 64-bit Windows platform, AnthillPro supports it with a 32-bit JVM; for other 64-bit platforms, AnthillPro uses a 64-bit JVM:

Operating System	JVM 32	JVM 64
Windows 32-bit	Yes	N/A
Windows 64-bit	Yes	No
Non-windows 32-bit	Yes	N/A
Non-windows 64-bit	Yes	Yes

Database Installation

If you are installing a production instance, you will most likely need to use a database other than Derby. Currently, AnthillPro supports the following databases to store information:

- Oracle. Use the JDBC driver for Oracle 10.2 or higher, regardless of what version of the database is used.

Starting with AnthillPro 3.7.3, you can use Oracle RAC (version 10g or later) as the backing database. For clean installs of AnthillPro, simply select Oracle as the backing database. If you are currently running AnthillPro with Oracle, switching to RAC is straightforward: after backing up your current database, update the base.xml and installed.properties files to use the new Oracle RAC configuration.

- MySQL with InnoDB storage (works with 4.1.22 and later). It is recommended that the InnoDB [<http://www.innodb.com/>] storage engine be used. Also recommended is to use **row or mixed binary logging format** (not statement).
- For MySQL 5, **use the 5.0.8 driver** version. The 5.1 version has known issues that will cause the AnthillPro server to throw an error. If you are using the 5.1 version, switch the driver jar file in the server's `lib/ext` directory and then restart.
- For MySQL 5, use the **5.0.8 connector**. There are some known issues in connector version 5.1.12 that result in errors.
- Microsoft SQL Server. The MS SqlServer 2.0 driver (`sqljdbc4.jar`) will not work with AnthillPro. To install the server using Microsoft SQL Server, you will need to use one of the following drivers, depending on which version of Java the AnthillPro server uses:
 - Java 5.** Use either the 1.2 driver or the 2.0 `sqljdbc.jar` driver.
 - Java 6.** Use the 1.2 driver.
- DB2 (requires 9.7 or later).
- PostgreSQL. Due to a known defect with PostgreSQL 8.4.0, AnthillPro can't be installed with that version. Use **version 8.3.7**.

Install Using DB2

AnthillPro uses the Apache Derby database by default, but can also use DB2 9.7 or later as a database. An existing installation of DB2 9.7 or later must be installed either on the same machine as the AnthillPro server (for evaluation purposes only) or somewhere accessible on the network. You will need to provide the AnthillPro installation with the connection information to the DB2 9.7 or later database as well as a user login that is allowed to create tables.

Once the DB2 9.7 or later Server database and user have been created, obtain the correct JDBC driver that allows AnthillPro to connect to the DB2 9.7 or later database.

1. Contact your DB2 9.7 or later vendor and request the JDBC driver that corresponds to the database version in use.
2. After downloading AnthillPro, expand the archive.
3. Copy the JDBC jar file into the **anthill3-install/lib/ext** directory.
4. Proceed with the normal installation instructions until you reach the choice of database. See either Linux/Unix Installation or Windows Installation.
5. When prompted for the database type, enter **db2**.
6. Provide the following:
 - **Driver.** The JDBC driver class that AnthillPro will use to connect to the database. Default: `com.ibm.db2.jcc.DB2Driver`.
 - **Connection string.** Format of the connection string that is dictated by the JDBC driver you are using. For example, `jdbc:db2://localhost:48665/anthill3`.

7. Follow the AnthillPro installation instructions for your operating system. See either Linux/Unix Installation or Windows Installation.

Install Using Oracle

AnthillPro uses the Apache Derby database by default, but can also use Oracle/Oracle RAC as a database. An existing installation of Oracle must be installed either on the same machine as the AnthillPro server (for evaluation purposes only) or somewhere accessible on the network. You will need to provide the AnthillPro installation with the connection information to the Oracle database as well as a user login that is allowed to create tables.

Starting with AnthillPro 3.7.3, you can use Oracle RAC (version 10g or later) as the backing database. For clean installs of AnthillPro, simply select Oracle as the backing database. If you are currently running AnthillPro with Oracle, switching to RAC is straightforward: after backing up your current database, update the `base.xml` and `installed.properties` files to use the new Oracle RAC configuration.

It is best to create a unique user for this database, and you may need to ensure that the user does not have Administrator privileges.

- Use the JDBC driver for Oracle 10.2 or higher, regardless of what version of the database is used.

To install the AnthillPro server with Oracle, download either the zip or tar.gz distribution. Once you have the distribution downloaded, you may unpack it, but do not start the installation yet.

Before installation, obtain a JDBC driver for the AnthillPro server to use. (*Use the JDBC driver for Oracle 10.2 or higher, regardless of what version of the database is used.*) The JDBC driver is the Java code that allows the server to connect to the Oracle database. Oracle should provide a JDBC jar file with its installation. The driver is unique to the version of Oracle you are running. You need to locate the JDBC jar file and copy it into the expanded installation directory.

The JDBC jar file needs to be copied to the **anthill3-install/lib/ext** directory.

Proceed with the normal installation instructions until you reach the choice of database. You will be prompted for the type of database you want to use. Enter `oracle`. You will also need to provide information specific to the Oracle database.

First, give the JDBC driver class that AnthillPro will use to connect to the database. The default value is **oracle.jdbc.driver.OracleDriver**. It's a good idea to check your Oracle installation or driver file to see if the class is different before allowing the default value.

Second, you will be prompted for the JDBC connection string. The format of the connection string is dictated by the JDBC driver. See Oracle documentation [<http://www.oracle.com/technology/documentation/index.html>].

Typically, it will match the following format:

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

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

- If you are having trouble, and have a DBA account, try including the service name parameter, similar to:

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

After providing the JDBC connection string, you will be prompted for the user and password. Enter those and complete the installation normally. If you receive errors during the installation, please copy the error output and contact support [<https://support.urbancode.com/>].

Install Using Microsoft SQL Server

AnthillPro can use Microsoft SQL server as its backing database. AnthillPro requires a newly created database in Microsoft SQL server, as well as a user that is allowed to create tables. It is best to install the database on a different machine as the server (except for evaluation purposes) under a unique user for AnthillPro to use.

You will also need to obtain the correct JDBC driver from Microsoft.

- The MS SqlServer 2.0 driver (sqljdbc4.jar) will not work with AnthillPro. To install the server using Microsoft SQL Server, you will need to use one of the following drivers, depending on which version of Java the AnthillPro server uses:

Java 5. Use either the 1.2 driver or the 2.0 sqljdbc.jar driver.

Java 6. Use the 1.2 driver.

First, create a database and a user in your SQL Server database before installing AnthillPro. You can do this by running the following commands (change the password to something secure):

```
CREATE DATABASE anthill3;
USE anthill3;
CREATE LOGIN anthill3 WITH PASSWORD = 'password';
CREATE USER anthill3 FOR LOGIN anthill3 WITH DEFAULT_SCHEMA = anthill3;
CREATE SCHEMA anthill3 AUTHORIZATION anthill3;
GRANT ALL TO anthill3;
```

Once the Microsoft SQL Server database and user have been created, obtain a JDBC driver for the AnthillPro server to use. The JDBC driver is the Java code that allows the server to connect to the Microsoft SQL Server database. On their web site, Microsoft provides a JDBC driver jar to download. Please download the driver version that corresponds to your version of Microsoft SQL Server.

- The MS SqlServer 2.0 driver (sqljdbc4.jar) will not work with AnthillPro. To install the server using Microsoft SQL Server, you will need to use one of the following drivers, depending on which version of Java the AnthillPro server uses:

Java 5. Use either the 1.2 driver or the 2.0 sqljdbc.jar driver.

Java 6. Use the 1.2 driver.

Locate the JDBC jar file and copy it into the AnthillPro installation directory at: anthill3-install/lib/ext.

Proceed with the normal installation instructions until you reach the choice of database. When prompted for the type of database to use, enter `sqlserver` to use the Microsoft SQL Server database. Also provide the JDBC driver class that AnthillPro will use to connect to the database. The default value is `com.microsoft.sqlserver.jdbc.SQLServerDriver`. Check your Microsoft SQL Server driver down-

load to see if the class is different before using the default.

Next, give the JDBC connection string. The format of the connection string is dictated by the JDBC driver. See Microsoft SQL Server documentation [<http://www.microsoft.com/sql/default.mspx>].

Typically, it will match the following format:

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

Eg. `jdbc:sqlserver://localhost:1433;databaseName=anthill3`

After providing the JDBC connection string, you will be prompted for the user and password. Enter those and complete the installation normally. If you receive errors during the installation, please copy the error output and contact support [<https://support.urbancode.com/>].

Install Using MySQL

AnthillPro uses the Apache Derby database by default, but you can also use MySQL (with InnoDB storage) as the database. An existing installation of MySQL (with InnoDB storage) must be installed either on the same machine as the AnthillPro server (if you are evaluating AnthillPro) or somewhere accessible on the network. You will need to provide the AnthillPro installation with the connection information to the MySQL database as well as a user login that is allowed to create tables. It is best to create a unique user for this database.

- MySQL with InnoDB storage (works with 4.1.22 and later). It is recommended that the InnoDB [<http://www.innodb.com/>] storage engine be used. Also recommended is to use **row or mixed binary logging format** (not statement).
- For MySQL 5, use the **5.0.8 driver** version. The 5.1 version has some bugs that will cause the AnthillPro server to throw an error. If you are using the 5.1 version, switch the driver jar file in the server's lib/ext directory and then restart.
- For MySQL 5, use the **5.0.8 connector**. There are some known issues in connector version 5.1.12 that result in errors.

To install the AnthillPro server with MySQL, download either the zip or tar.gz distribution. Once you have the distribution downloaded, unpack it but do not start the installation yet.

First, create a database and a user in your MySQL database before installing AnthillPro. You can do this by running the following commands (change the password to something secure):

```
CREATE DATABASE anthill3;

GRANT ALL ON anthill3.* TO 'anthill3'@'%'
IDENTIFIED BY 'password' WITH GRANT OPTION;
```

Before installation, obtain a JDBC driver for the AnthillPro server to use. The JDBC driver is the Java code that allows the server to connect to the MySQL database. MySQL should provide a JDBC jar file with its installation. The driver is unique to the version of MySQL you are running. Locate the JDBC jar file and copy it into the expanded installation directory. The JDBC jar file needs to be copied to the `anthill3-install/lib/ext` directory.

Proceed with the normal AnthillPro installation instructions until you reach the choice of database. You will be prompted for the type of database you want to use, so enter `mysql`. You will also need to provide need to provide the JDBC driver class that AnthillPro will use to connect to the database. The default value is `com.mysql.jdbc.Driver`. Check your MySQL installation or driver file to see if the class is different before

accepting the default value.

Next, you will be prompted for the JDBC connection string. The format of the connection string is dictated by the JDBC driver. See MySQL documentation [<http://dev.mysql.com/doc/>].

Typically, it will match the following format:

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

Eg. `jdbc:mysql://localhost:3306/anthill3`

After providing the JDBC connection string, you will be prompted for the user and password. Enter those and complete the installation normally. If you receive errors during the installation, please copy the error output and contact support [<https://support.urbancode.com/>].

Install Using PostgreSQL

- Due to a known defect with PostgreSQL 8.4.0, AnthillPro can't be installed with that version. Use **version 8.3.7**.

AnthillPro uses the Apache Derby database by default, but can also use PostgreSQL as a database. An existing installation of PostgreSQL must be installed either on the same machine as the AnthillPro server (for evaluation purposes only) or somewhere accessible on the network. You will need to provide the AnthillPro installation with the connection information to the PostgreSQL database as well as a user login that is allowed to create tables.

Once the PostgreSQL Server database and user have been created, obtain the correct JDBC driver that allows AnthillPro to connect to the PostgreSQL database.

1. Go to [www.PostgreSQL.org](http://jdbc.postgresql.org/download.html) and download the JDBC driver [<http://jdbc.postgresql.org/download.html>] for the version of PostgreSQL you use.
2. After downloading AnthillPro, expand the archive.
3. Copy the JDBC jar file into the **anthill3-install/lib/ext** directory.
4. Proceed with the normal installation instructions until you reach the choice of database. See either Linux/Unix Installation or Windows Installation.
5. When prompted for the database type, enter **postgres**.
6. Provide the following:
 - **Driver.** The JDBC driver class that AnthillPro will use to connect to the database. Default: `org.postgresql.Driver`.
 - **Connection string.** Format of the connection string that is dictated by the JDBC driver you are using. For example, `jdbc:postgresql://localhost:5432/anthill3`.
7. Follow the AnthillPro installation instructions for your operating system. See either Linux/Unix Installation or Windows Installation.

If you receive errors during the installation, please copy the error output and contact support [support.urbancode.com].

Download AnthillPro

The installation package is available from the Urbancode Support Portal. If you are evaluating AnthillPro, the Supportal account where you download AnthillPro also gives you access to the support team -- where you can create a support ticket.

1. Go to the Urbancode Support Portal [<http://support.urbancode.com/>]. You must already be a license holder to access Supportal. If you do not have an account, please create one.
2. Go to the **PRODUCTS** tab and select the version of AnthillPro you are downloading.
3. Select the appropriate command-line installer. The contents of both packages are the same:
 - **anthill3-<version>.tar.gz**. Command line installer for Unix/Linux.
 - **anthill3-<version>.zip**. Command line installer for Windows.

AnthillPro allows you to install Agents on any supported platform, regardless of the operating system the server is installed on. To do so, simply run the appropriate script included in the install package you chose. See either Installing the Agent (Windows) or Installing the Agent (Linux/Unix).

4. See either Windows Installation or Linux/Unix Installation.

Windows Installation

To complete an installation, yinstall the server and at lest one agent. Typically, an agent is installed on a different machine as the server to avoid performance issues. When the server and agent are on the same machine, a busy build environment can slow down the server.

For evaluation purposes, installing the agent and server on the same machine should suffice. If you do this, it is best to install the agent in its own directory (e.g., c:\anthill-agent) and not in the server's directory.

Before you continue, ensure that you have the correct JVM/JDK for the server and agent. (It is recommended to use the Sun JDK. There have been some issues with the third-party JDK's from IBM and others.) If you are using a database other than Derby, make sure you have reviewed the Installation section and have gathered the appropriate driver, etc.

Server Installation (Windows)

For newer versions of Windows, you will need to execute an Administrator Command Prompt to install as a Windows Service. This can be done by right clicking `cmd.exe` and selecting "Run As Administrator."

To install the AnthillPro server:

1. Download the **anthill3-<version>.zip** file.
2. Expand the zip file using a tool like WinZip. Expanding will create an **anthill3-install** directory.
3. Open the **anthill3-install** directory created in the previous step in **Windows**.
4. If you are using a database other than Apache Derby, copy the JDBC driver file(s) into the **anthill3-install\lib\ext** directory. See Database Requirements.

5. Run the install script **install-server.bat**.
6. Provide the following information:
 - **Directory** where the AnthillPro **server** should be installed. For example: **C:\Program Files\anthill3\server**. If the directory does not already exist, enter **Y** to have the installer create the directory. Default is **Y**.
 - **Home directory of the JRE/JDK** used to run the server. For example: **C:\Program Files\Java\jre1.6.0_07**.
 - **Port** used to communicate with the agent(s). Recommended default value is **7915**.
 - **Port** on which AnthillPro server should listen for remoting or distributed servers. Default is **4567**.
 - **IP address** that the server web UI should **listen on**. We suggest leaving this blank so that AnthillPro listens on **all available IP addresses**.
 - **Secured connections using SSL**. If using a secured connection for all web UI communication, enter **Y**.

*When using SSL, you need to turn it on for both the **server and the agent** -- otherwise the agents will not be able to connect to the server. If you are using the agent relay, ensure that **SSL is turned on for all three components: server, agent, and agent relay**. This rule also applies if using mutual authentication.*

If using a caching proxy, see Caching and SSL before continuing. Under certain conditions, using HTTPS may effect server performance.

- If using a secured connection for HTTP requests, give the port the AnthillPro sever should listen on. Recommended default is **8443**.
 - **Unsecured HTTP port for the Web UI**. This will be used to redirect unsecured requests to the secured HTTPS port when using HTTPS for the Web UI. Default is **8080**.
 - **HTTP port for the Web UI** (if not using SSL). Default is **8080**.
 - **Database type** (if other than the embedded Derby database). See Database Requirements.
 - **Database user name**. Default is **anthill3**.
 - **Database password**. Default is **password**.
7. **Install as Windows service (optional)**. Type **Y** to install as Windows service. Type **N** to finish installation process. Note that when installing as a service, AnthillPro only captures the value for the PATH variable. The values captured at installation will always be used, even if you later make changes.

For newer versions of Windows, you will need to execute an Administrator Command Prompt to install as a Windows Service. This can be done by right clicking `cmd.exe` and selecting "Run As Administrator."

8. To **automatically install** the AnthillPro server as a Windows service, **enter**:
 - **Service name** and click **OK**. Default is **ah3server**. (*A unique name is required for each instance.*)
 - **Login** for the installed server and click **OK**. Default is **.\localsystem**.
 - **Yes** to automatically start ah3server service.

To manually install the AnthillPro server as a Windows service (optional):

For newer versions of Windows, you will need to execute an Administrator Command Prompt to install as a Win-

dows Service. This can be done by right clicking `cmd.exe` and selecting "Run As Administrator."

- Run **ah3server.cmd install uniqueservicename** (located in the server's **bin\service** directory).
- Provide the **service name** and **account information**.

To run the AnthillPro server and complete installation:

1. In Windows, go to the **AnthillPro server directory** created during installation. For example, **C:\Program Files\anthill3\server**. Enter the **bin** directory.
2. Run: **start_ah3server.cmd**.
3. Alternatively, from the **command line** run the start command **bin\ah3server.cmd**.
4. When the AnthillPro server has started, open a web browser and give the AnthillPro URL. For example, **http://localhost:8080**.
5. Complete installation:
 - **External URL.** Enter the URL users will use to access AnthillPro. This URL will be used to generate links back to the application in e-mail notifications, as well as for some agent communication. The URL may include **http(s)://** and any non-standard ports. For example **http://localhost:8080** if you are using default installation values.
 - **Administrator E-mail Address.** Give the e-mail address that should be notified about system issues. The Administrator account will have complete access to AnthillPro, and is the initial user that must set up security. See Security.
 - **Administrator Password.** Enter the password the AnthillPro administrator (created above) will use to access AnthillPro.

Note that this password (and the user name **admin**) will be required to complete installation of the Distributed Web interface if the AnthillPro server is going to be used in conjunction with the Distributed Web product. For information on the Distributed Web interface, contact <info@urbancode.com>.

- **Confirm Administrator Password.** Enter the password again.
- **License.** Paste your AnthillPro license in the text field after retrieving it from Supportal [<http://www.support.urbancode.com/>].
 - a. Go to Supportal [<http://www.support.urbancode.com/>] to retrieve the license from your account.
 - b. Go to **TEAMS/USERS > Licenses**. If you do not see a license, either one does not exist for your account or you do not have permissions to download a license. Please contact your sales representative for more information.
 - c. Select the **view license** link on the right hand side of the appropriate license.
 - d. In the pop-up, click **download**.
 - e. Open the license, copy it, and paste it in the **License** field.

If you are evaluating AnthillPro, you download a 30-day, temporary license. The only difference between this and a production license is the expiration date. If you need a new production license, please contact your sales represent-

ative or create a Supportal [<http://www.support.urbancode.com/>] ticket with your request.

6. Click **Complete**.

The application can now be accessed in the web browser. If you log out, give the user name **admin** and the password set in Item Five to log back in.

7. **Select pre-installed Plugins to activate/deactivate.** AnthillPro ships with pre-configured integrations that are written as Plugins. If you choose to deactivate any of the Plugins, you will not be able to use the AnthillPro integration with that tool until you reactivate it. If you don't know what tool integrations you want deactivated, you can leave them all active to start with and then edit the setting later. Click **Done** to go to the Dashboard.

Installing the Agent (Windows)

When installing the agent in a *production* environment, most people typically create a dedicated user that is responsible for running the agent (i.e., they create an "Anthill" user on the machine the agent is being installed on).

- Note that for evaluation purposes, you can simply run the agent as the "Admin" user.

In general, the agent will need the appropriate rights in order to communicate with the AnthillPro server; to communicate with any tools AnthillPro will integrate with (e.g., repository, testing tool, issue tracking tool, e-mail/IM, etc.); and to perform other tasks you will want AnthillPro to perform. Each agent should have permission to:

- **Create a cache.** By default the cache is located in the home directory of the user running AnthillPro. However, the cache may be moved or disabled.
- **Open a TCP connection.** The agent must use a TCP connection to communicate with the AnthillPro server's JMS port.
- **Open a HTTP(S) connection.** The agent must be able to connect to the AnthillPro UI to download artifacts from the AnthillPro artifact repository (Codestation).
- **Access the file system.** Many agents will need read/write permissions to items on the file system that are expected to go through AnthillPro.

To install the AnthillPro agent:

1. Download the **anthill3-<version>.zip** file.
2. Expand the zip file using a tool like WinZip. Expanding will create an **anthill3-install** directory.
3. Open the **anthill3-install directory** created in the previous step in **Windows**.
4. Run the install script **install-agent.bat**. For advanced agent installation, see also Unattended Agent Installation Scripts.
5. Provide the following information:
 - **Directory** where the AnthillPro **agent** should be installed. For example: **C:\Program Files\anthill3\agent**. To have the installer create the directory, enter **Y**.

- Home directory of the **JRE/JDK** used to run the agent. Default is **C:\Program Files\Java\jre1.6.0_07**.
- **Relay connection.** Enter the default **N** if *not* using UrbanCode's Distributed Servers product, available under separate license. If using Distributed Servers, see Installing Distributed Servers before continuing.

If you are interested in using the Agent Relay for agent-server communication, contact <info@urbancode.com>.

- Enter the **host name** or **address** of the AnthillPro server. Default is **localhost**.
- Enter the **port** which the AnthillPro server will use for internal communication. Default is **7915**.
- Determine if internal communication should use **SSL**. Default is **N**.

*When using SSL, you need to turn it on for both the **server and the agent** -- otherwise the agents will not be able to connect to the server. If you are using the agent relay, ensure that **SSL is turned on for all three components: server, agent, and agent relay**. This rule also applies if using mutual authentication.*

- If using SSL, determine if communication should be **mutually authenticated**. Default is **N**.

6. Enter name of **agent**.

7. Type **Y** to automatically install the AnthillPro agent as a Windows service. Type **N** to complete installation process.

For newer versions of Windows, you will need to execute an Administrator Command Prompt to install as a Windows Service. This can be done by right clicking `cmd.exe` and selecting "Run As Administrator."

8. **Install Agent as Windows Service.** Note that when installing as a service, AnthillPro only captures the value for the **PATH** variable. The values captured at installation will always be used, even if you later make changes.

For newer versions of Windows, you will need to execute an Administrator Command Prompt to install as a Windows Service. This can be done by right clicking `cmd.exe` and selecting "Run As Administrator."

If automatically installing the AnthillPro agent as a Windows service, enter:

- Service **unique name**. Default is **ah3agent**. (A unique name is required for each instance.)
- **Login** for installed agent service. Default is **.\localsystem**.
- **Yes** to automatically start service.

To manually install the AnthillPro agent as a Windows service:

- Run **ah3agent.cmd install uniqueservicename** (located in **bin\service** folder of the AnthillPro directory).
- Provide the service **name** and **account information**.

To run the AnthillPro agent:

1. In Windows, navigate to the **AnthillPro agent directory** created during the installation. Default is: **C:\Program Files\anthill3\agent**. Enter the **bin** directory.

2. Run the start script: **start_ah3agent.cmd**.
3. Alternatively, from the **command line** run the start command **bin\ah3agent.cmd start**.
4. See Configure Agents. You will not be able to build a project until an agent is properly configured.

Linux/Unix Installation

To complete an installation, install the server and at least one agent. Typically, an agent is installed on a different machine as the server because the agent is responsible for performing resource-intensive tasks such as builds. When the server and agent are on the same machine, a busy build environment can effect server performance.

For evaluation purposes, installing the agent and server on the same machine should suffice. If you do this, it is best to install the agent in its own directory.

Server Installation (Linux/Unix)

To install the AnthillPro server:

1. Download the **anthill3-<version>.tar.gz** file.
2. Open a **UNIX shell** to the directory containing the above downloaded file.
3. Extract the downloaded file. Type: **tar -zxf anthill3-<version>.tar.gz**.
 - On some installations of Solaris and HP-UX the default tar command will not properly handle our tar files. You may need to use `\ install GNU tar`.

When installing AnthillPro on Solaris, it is recommended to use korn shell (ksh).
4. **cd anthill3-install**.
5. If you are using a database other than Apache Derby, copy the JDBC driver file(s) into the **anthill3-install/lib/ext** directory. See Database Requirements.
6. Run the install script **./install-server.sh**.

You can also install the server to run automatically using the init.d script. To do so, go to the install directory: `/anthill3-install/bin/server/init`, copy the `ah3server` file into your `init` directory, and make sure the `INSTALL MODIFICATION` section contains the correct information. Once your modifications are completed, make the file available to run.

7. Provide the following:
 - **Directory** where the AnthillPro **server** should be installed. For example: `/opt/anthill3/server`. If the directory does not already exist, enter **Y** to have the installer create the directory. Default is **Y**.
 - **Home directory of the JRE/JDK** used to run the server.
 - **Port** on which AnthillPro server should listen for remoting or distributed servers. Default is **4567**.
 - **IP address** that the server web UI should **listen on**. We suggest using the default 'all IP addresses bound to machine' so that AnthillPro listens on **all available IP addresses**.

- **JMS port** of the server. This is the port used for internal communications between the agent(s) and server. Recommended default value is **7915**.
- **Secured connections using SSL**. If using a secured connection for all web UI communication, enter **Y**. If using a caching proxy, see Caching and SSL before continuing. Under certain conditions, using HTTPS may affect server performance.

*When using SSL, you need to turn it on for both the **server and the agent** -- otherwise the agents will not be able to connect to the server. If you are using the agent relay, ensure that **SSL is turned on for all three components: server, agent, and agent relay**. This rule also applies if using mutual authentication.*

- If using a secured connection for HTTP requests, give the port the AnthillPro sever should listen on. Recommended default is **8443**.
- **Unsecured HTTP port for the Web UI**. This will be used to redirect unsecured requests to the secured HTTPS port when using HTTPS for the Web UI. Default is **8080**.
- **HTTP port for the Web UI** (if not using SSL). Default is **8080**.
- **Database type** (if other than the embedded Derby database). See Database Requirements.
- **Database user name**. Default is **anthill3**.
- **Database password**. Default is **password**.

To run the AnthillPro server and complete installation:

1. Open a **UNIX shell** to the directory where you installed the server. If you took our suggestion, then this is: **/opt/anthill3/server**.
2. **cd bin**.
3. Run the start script: **./ah3server start**.
4. The application can now be accessed in the web browser.
5. Complete installation:
 - **External URL**. Enter the URL users will use to access AnthillPro. This URL will be used to generate links back to the application in e-mail notifications, as well as for some agent communication. The URL may include **http(s)://** and any non-standard ports. For example **http://localhost:8080** if you are using default installation values.
 - **Administrator E-mail Address**. Give the e-mail address that should be notified about system issues. The Administrator account will have complete access to AnthillPro, and is the initial user that must set up security. See Security.
 - **Administrator Password**. Enter the password the AnthillPro administrator (created above) will use to access AnthillPro.

Note that this password (and the user name **admin**) will be required to complete installation of the Distributed Web interface if the AnthillPro server is going to be used in conjunction with the Distributed Web product. For information on the Distributes Web interface, contact <info@urbancode.com>.

- **Confirm Administrator Password**. Enter the password again.

- **License.** Paste your AnthillPro license in the text field after retrieving it from Supportal [<http://www.support.urbancode.com/>].
 - a. Go to Supportal [<http://www.support.urbancode.com/>] to retrieve the license from your account.
 - b. Go to **TEAMS/USERS > Licenses**. If you do not see a license, either one does not exist for your account or you do not have permissions to download a license. Please contact your sales representative for more information.
 - c. Select the **view license** link on the right hand side of the appropriate license.
 - d. In the pop-up, click **download**.
 - e. Open the license, copy it, and paste it in the **License** field.

If you are evaluating AnthillPro, you download a 30-day, temporary license. The only difference between this and a production license is the expiration date. If you need a new production license, please contact your sales representative or create a Supportal [<http://www.support.urbancode.com/>] ticket with your request.

6. Click **Complete**.

The application can now be accessed in the web browser. If you log out, give the user name **admin** and the password set in Item Five to log back in.

7. **Select pre-installed Plugins to activate/deactivate.** AnthillPro ships with pre-configured integrations that are written as Plugins. If you choose to deactivate any of the Plugins, you will not be able to use the AnthillPro integration with that tool until you reactivate it. If you don't know what tool integrations you want deactivated, you can leave them all active to start with and then edit the setting later. Click **Done** to go to the Dashboard.

Installing the Agent (Linux/Unix)

When installing the agent in a *production* environment, most people typically create a dedicated user that is responsible for running the agent (i.e., they create an "Anthill" user on the machine the agent is being installed on).

- Note that for evaluation purposes, you can simply run the agent as the "Admin" user.

In general, the agent will need the appropriate rights in order to communicate with the AnthillPro server; to communicate with any tools AnthillPro will integrate with (e.g., repository, testing tool, issue tracking tool, e-mail/IM, etc.); and to perform other tasks you will want AnthillPro to perform. Each agent should have permission to:

- **Create a cache.** By default the cache is located in the home directory of the user running AnthillPro. However, the cache may be moved or disabled.
- **Open a TCP connection.** The agent must use a TCP connection to communicate with the AnthillPro server's JMS port.
- **Open a HTTP(S) connection.** The agent must be able to connect to the AnthillPro UI to download artifacts from the AnthillPro artifact repository (Codestation).
- **Access the file system.** Many agents will need read/write permissions to items on the file system that are expected to go through AnthillPro.

See Agent Installation (VMS) if you want to install the agent for VMS. The process is manual, and uses an existing, but modified, UNIX-installation.

To install the Agent:

1. Download the **anthill3-<version>.tar.gz** file.
2. Open a **UNIX shell** to the directory containing the above downloaded file.
3. Extract the downloaded file. Type: **tar -zxf anthill3-<version>.tar.gz**.
 - On some installations of Solaris and HP-UX the default tar command will not properly handle our tar files. You may need to use `\ install GNU tar`.

When installing AnthillPro on Solaris, it is recommended to use korn shell (ksh).

4. cd **anthill3-install**.
5. Run the install script **./install-agent.sh**.

You can also install the agent to run automatically using the init.d script. To do so, go to the install directory: `/anthill3-install/bin/agent/init`, copy the `ah3agent` file into your `init` directory, and make sure the `INSTALL MODIFICATION` section contains the correct information. Once your modifications are completed, make the file available to run.

6. Provide the following information:
 - **Directory** where the AnthillPro **agent** should be installed. For example: **/opt/anthill3/agent**. To have the installer create the directory, enter **Y**.
 - Home directory of the **JRE/JDK** used to run the agent.
 - **Relay connection**. Enter the default **N** if *not* using UrbanCode's Distributed Servers product, available under separate license. If using Distributed Servers, see Installing Distributed Servers before continuing.

If you are interested in using the Agent Relay for agent-server communication, contact `<info@urbancode.com>`.
 - Enter the **host name** or **address** of the AnthillPro server. Default is **localhost**.
 - Enter the **port** which the AnthillPro server will use for internal communication. Default is **7915**.
 - Determine if internal communication should use **SSL**. Default is **N**.
 - If using SSL, determine if communication should be **mutually authenticated**. Default is **N**.
7. Enter name of **agent**.

To run the AnthillPro agent, follow the steps below:

1. Open a **UNIX shell** to the directory where you installed the agent. If you took our suggestion, then this is: **/opt/anthill3/agent**.
2. cd **bin**.

3. Run the script: `./ah3agent start`.
4. See Configure Agents. You will not be able to build a project until an agent is properly configured.

Configure SSL

AnthillPro allows the server and agents to communicate securely using SSL in two modes: unauthenticated and mutual authentication. In unauthenticated mode, the communication between the server and agents is secured with SSL, but there is no authentication to verify the server or agent credentials.

*When using SSL, turn it on for both the **server and the agent** -- otherwise the agents will not be able to connect to the server. If you are using the agent relay, ensure that **SSL is turned on for all three components**: server, agent, and agent relay. This rule also applies if using mutual authentication.*

In mutual authentication mode, the communication between server and agent is secured with SSL and authentication takes place to verify the server is really the server and the agent is really the agent.

Configuring Server-Agent SSL

To use SSL between the server and agent, both the sides need a private key created in a keystore. This key and keystore is created during the installation process of the server and agent. There should be a `ah3.keystore` file in the `conf` directory of each installation. If there is not, please contact support [<https://support.urbancode.com/>] for assistance.

To turn SSL on, make sure the server is running and go to the Web UI. On the **System** tab, click the **Server Settings** link under the Server menu. In the new window, enable the **Use SSL Between Server and Agents** setting. After this, restart the server. That completes the server configuration for enabling SSL.

*When using SSL, turn it on for both the **server and the agent** -- otherwise the agents will not be able to connect to the server. If you are using the agent relay, ensure that **SSL is turned on for all three components**: server, agent, and agent relay. This rule also applies if using mutual authentication.*

For each agent installed, first shutdown the agent, and then go to the `conf/agent` directory and edit the agent's `installed.properties` file (note: this file was previously `agent.properties` on older installations). The settings in the file that controls the SSL communication is `locked/server.secure`. If this property is not present add the line: `locked/server.secure=Y`.

Start the agent, and go in to the server Web UI to verify that the agent shows up as online. To further test that communication is working properly, go to the **Agent** and select the **Properties** tab to validate the agent properties are retrieved correctly.

Configuring Mutual Authentication

For mutual authentication between server and agent communication, you must facilitate a public key exchange between the two. To do this, export the public key from the certificate keystore and import it into the keystore of that which you are communicating with: Export the server key and import it into the agent keystore, and export the agent key and import it into the server keystore. Also ensure that every agent using mutual authentication has the `locked/server.mutual_auth=Y` property in the `installed.properties` (or `agent.properties`) file.

1. To export the public key from the server certificate keystore, open a command-line shell to the server installation's `conf` directory and run:

```
keytool -export -keystore ah3.keystore -storepass changeit  
-alias ah3_server -file ah3_server.crt
```

2. This file needs to be copied to your agents. Place it into the agent installation's conf directory and run the following to import it:

```
keytool -import -keystore ah3.keystore -storepass changeit  
-alias ah3_server -file ah3_server.crt -keypass changeit -noprompt
```

3. While on the agents, run the following from their installation's conf directory to export their public keys (change the name of the file argument to match the agent name):

```
keytool -export -keystore ah3.keystore -storepass changeit  
-alias ah3_agent -file [agent_name].crt
```

4. Open the Agent's installed.properties (or agent.properties) file (in the conf\agent directory) in a text editor. Set the locked/server.mutual_auth property to true: locked/server.mutual_auth=Y and save your changes.

5. Copy the agent public keys to the server's conf directory and run the following command to import them (change the name of the file argument to match your file name and the name of the alias argument to match your agent name):

```
keytool -import -keystore ah3.keystore -storepass changeit  
-alias [agent_name] -file [agent_name].crt -keypass changeit -noprompt
```

6. To complete mutual authentication, make sure the server is running, go to the **System** page, and click the **Server Settings** link under the Server menu.
7. In the next window, enable the **Enforce Mutual Authentication** setting.
8. Restart agent(s).
9. Verify that communication is working properly between server and agents.

To list the certificates that are loaded into a keystore, run the following from the same directory:

```
keytool -list -keystore ah3.keystore -storepass changeit
```

SSL and Distributed Servers

It is possible to configure the AnthillPro server to use SSL with the Distributed Web Interface. To do so, import an existing certificate and switch to https. Before you begin, make sure AnthillPro is not running any jobs (you can do this on the **Current Activity** page). The ports listed in the examples may be different than those used by your AnthillPro system. See Using SSL with Distributed Web Interface.

Agent Installation (VMS)

AnthillPro Agent installation must be done manually for VMS, as there are no installation scripts included with the product. This is easiest if an existing Unix-based installation is modified and packaged for installation on the VMS system.

The only prerequisite for the target VMS system:

- JVM installed must be at least Java 1.4.
- The logical `JAVA$FORK_PIPE_STYLE` must be set to 1.

There are no modifications to the VMS environment as a result of the agent installation.

To install:

1. Choose any existing Unix-based agent installation and make a complete copy.

- Run: `cp -R agent vmsAgent`

2. Modify `classpath.conf` file.

- Run: `vi vmsAgent/bin/classpath.conf`

- Change all paths in file to match the target VMS path. VMS paths must be specified using Unix-style as opposed to VMS-style. For example:

```
dir /opt/anthill3/agent/conf/client
```

becomes

```
dir /VSI_KIT_ROOT1/ANTHILL/KIT/AGENT/CONF/CLIENT
```

3. Modify `agent.properties` file.

- Run: `vi vmsAgent/conf/agent/agent.properties`

- Change all paths in file to match the target VMS path. VMS paths must be specified using Unix-style as opposed to VMS-style. For example:

```
anthill3/temp.dir=/opt/anthill3/agent/var/temp
```

becomes

```
anthill3/temp.dir=/VSI_KIT_ROOT1/ANTHILL/KIT/AGENT/VAR/TEMP
```

- The following is a list of all path properties that need to be changed.

```
anthill3/temp.dir
```

```
anthill3/var.dir
```

```
locked/agent.home
```

```
locked/java.home
```

```
anthill3/work.dir
```

```
anthill3/logs.dir
```

```
anthill3/root.dir
```

```
locked/ant.home
```

- Change the agent name property (replace {VMS host name} with the actual host name).

```
locked/anthill3.name <http://anthill3.name>={VMS host name}
```

- Change the agent ID property. The ID can be anything as long as it's unique from every other agent (host name may be used if desired).

```
locked/anthill3.id <http://anthill3.id>={desired ID}
```

4. Create a VMS DCL file to enable launching of the agent process.

- Run: `vi vmsAgent/bin/AH3AGENT.COM <http://AH3AGENT.COM>`

- Add the following content to the file.

```
$ ! setup.com <http://setup.com>
$
$ @sys$startup:java$150_setup.com <http://150_setup.com>
$
$ java -jar [.]launcher.jar [.]classpath.conf
$ "com.urbancode.anthill3.main.agent.AnthillAgent"
$
```

5. Package the VMS agent installation.

Run:

```
cd vmsAgent zip -r anthill_vms_agent.zip *
```

- Run: `cd vmsAgent zip -r anthill_vms_agent.zip *`

6. Copy the agent package to the target VMS host.

7. Unzip the agent package in desired location (must match paths used in steps 2 and 3).

8. Start the agent.

- Run:

```
SET DEFAULT VSI_KIT_ROOT1:[ANTHILL.KIT.AGENT.BIN]
@AH3AGENT
```

9. Open a browser and go to `http://anthill.unix.swx.ch <http://anthill.unix.swx.ch/>`

- Log in as "admin" using password from Server Installation.
- Click the **System** tab. In some versions of AnthillPro, agents have their own top-level **Agents** tab.
- Click Agents in the **Environment Administration** section.
- Find the new agent under Available Agents.
- Click **Configure**.
- Select the Build-Farm checkbox.
- Click **Set** then **Done**.

Unattended Agent Installation Scripts

To install a large number of agents quickly, AnthillPro provides an Unattended Agent Installation script that invokes the agent installer. The script allows you to specify how many agents to install in a chosen directory, and set the requisite agent properties. The script uses all of the agent installation defaults, so you will most likely need to modify the script to fit your needs. Once the script is modified, it is also possible to customize it; e.g., install agents in different directories, install agents with different names, etc.

The Unattended Agent Installation script is available with the standard AnthillPro installation package. There are two versions of the script, one for installing agents on Windows and one for installing agents on Linux/Unix.

Unattended Agent Installation Script (Windows)

Below is a copy of `unattended-install-agent.bat`, which is included in the AnthillPro Installation Package. The script invokes the AnthillPro agent installer, and requires you to set all the agent properties that are given during a command-line installation. To get started:

1. Download the appropriate AnthillPro Installation Package.
2. Open `unattended-install-agent.bat` file in a text editor.
3. **Set the agent properties.** Please refer to Installing the Agent (Windows) and the AnthillPro server `install.properties` file (located in the `\conf\server\` directory) for the appropriate settings.
4. Once the properties are set, *remove lines 11 and 12*, which prevent the script from accidentally running. The easiest way to do this is simply comment out the lines as follows:

```
REM echo REMOVE THIS LINE AFTER MODIFYING THIS SCRIPT FOR YOUR NEEDS
REM GOTO END
```

- Note that if the `unattended-install-agent.bat` script is not in the same directory as the `install-agent.bat` file, you will have to make additional modifications to script to install the agent(s).

5. Save your changes.
6. Open the command line, point it to the **anthill3-install** directory, and run `unattended-install-agent.bat`.
7. Start the agent(s). See Installing the Agent (Windows).

Example Unattended Agent Installation Script (Windows):

```
@echo off
setlocal
REM #####
REM WARNING: This script is an example of how to create an unattended
REM installation script. The parameters below as well as the script content
REM may and probably WILL need to be modified to accommodate your situation.
REM
REM The parameters which can be modified to alter the unattended installation.
REM #####

echo REMOVE THIS LINE AFTER MODIFYING THIS SCRIPT FOR YOUR NEEDS
GOTO END

set AGENT_JAVA_HOME=C:\Program Files\Java\jre6
```

```

set CONNECT_VIA_RELAY=N
set INSTALL_AGENT_REMOTE_HOST=<PUT ANTHILL3 SERVER HOSTNAME HERE>
REM IF CONNECT_VIA_RELAY
REM set INSTALL_AGENT_REMOTE_PORT=7916
REM ELSE
set INSTALL_AGENT_REMOTE_PORT=7915

set INSTALL_AGENT_REMOTE_PORT_SSL=N
set INSTALL_AGENT_REMOTE_PORT_MUTUAL_AUTH=N
REM IF CONNECT_VIA_RELAY
set INSTALL_AGENT_RELAY_HTTP_PORT=20080
REM ELSE
REM set INSTALL_AGENT_RELAY_HTTP_PORT=

set INSTALL_AGENT_NAME=anthill3-agent
set INSTALL_AGENT_DIR=C:\Program Files\anthill3\batch\agent

set INSTALL_SERVICE=N
set INSTALL_SERVICE_NAME=ah3agent
set INSTALL_SERVICE_LOGIN=.\localsystem
set INSTALL_SERVICE_PASSWORD=nopass
set INSTALL_SERVICE_AUTOSTART=N

set agent_count=10

REM #####
REM The installation script
REM #####

cd %~dp0

set ANT_HOME=opt\apache-ant-1.7.1
set CLASSPATH=

set i=0
:LOOP
CALL opt\apache-ant-1.7.1\bin\ant.bat -f install.with.groovy.xml ^
  "-Dinstall-agent=true" ^
  "-Dinstall-server=false" ^
  "-Dinstall.java.home=%AGENT_JAVA_HOME%" ^
  "-Dinstall.agent.connect_via_relay=%CONNECT_VIA_RELAY%" ^
  "-Dinstall.agent.jms.remote.host=%INSTALL_AGENT_REMOTE_HOST%" ^
  "-Dinstall.agent.jms.remote.port=%INSTALL_AGENT_REMOTE_PORT%" ^
  "-Dinstall.agent.relay.http.port=%INSTALL_AGENT_RELAY_HTTP_PORT%" ^
  "-Dinstall.agent.remote.port.ssl=%INSTALL_AGENT_REMOTE_PORT_SSL%" ^
  "-Dinstall.agent.remote.port.mutual_auth=%INSTALL_AGENT_REMOTE_PORT_MUTUAL_AUTH%" ^
  "-Dinstall.agent.name=%INSTALL_AGENT_NAME%-i%" ^
  "-Dinstall.agent.dir=%INSTALL_AGENT_DIR%-i%" ^
  "-Dinstall.service=%INSTALL_SERVICE%" ^
  "-Dinstall.service.name=%INSTALL_SERVICE_NAME%" ^
  "-Dinstall.service.login=%INSTALL_SERVICE_LOGIN%" ^
  "-Dinstall.service.password=%INSTALL_SERVICE_PASSWORD%" ^
  "-Dinstall.service.autostart=%INSTALL_SERVICE_AUTOSTART%" ^
  install-non-interactive
set /a i=i+1
if "%i%"=="%agent_count%" goto END
goto LOOP
:END

```

Unattended Agent Installation Script (Linux/Unix)

Below is a copy of unattended-install-agent.sh, which is included in the AnthillPro Installation Pack-

age. The script invokes the AnthillPro agent installer, and requires you to set all the agent properties that are given during a command-line installation. To get started:

1. Download the appropriate AnthillPro Installation Package.
2. Open `unattended-install-agent.sh` file in a text editor.
3. **Set the agent properties.** Please refer to `Installing the Agent (Linux/Unix)` and the `AnthillPro server install.properties` file (located in the `/conf/server/` directory) for the appropriate settings.
4. Once the properties are set, *remove lines 9 and 10*, which prevent the script from accidentally running. The easiest way to do this is simply comment out the lines as follows:

```
# echo REMOVE THIS LINE AFTER MODIFYING THIS FILE FOR YOUR PURPOSES
# exit
```

- Note that if the `unattended-install-agent.sh` script is not in the same directory as the `install-agent.sh` file, you will have to make additional modifications to script to install the agent(s).

5. Save your changes.
6. Open the command line, point it to the `anthill3-install` directory, and run `unattended-install-agent.sh`.
7. Start the agent(s). See `Installing the Agent (Linux/Unix)`.

Example Unattended Agent Installation Script (Linux/Unix):

```
#!/bin/sh
#####
# WARNING: This script is an example of how to create an unattended
# installation script. The parameters below as well as the script content
# may and probably WILL need to be modified to accommodate your situation.
#
# The parameters which can be modified to alter the unattended installation.
#####
echo REMOVE THIS LINE AFTER MODIFYING THIS FILE FOR YOUR PURPOSES
exit

AGENT_JAVA_HOME=/usr/lib/jvm/java-6-sun-1.6.0.10

CONNECT_VIA_RELAY=N
INSTALL_AGENT_REMOTE_HOST=<YOUR SERVER HOSTNAME HERE>
# IF CONNECT_VIA_RELAY
#INSTALL_AGENT_REMOTE_PORT=7916
# ELSE
INSTALL_AGENT_REMOTE_PORT=7915

INSTALL_AGENT_REMOTE_PORT_SSL=N
INSTALL_AGENT_REMOTE_PORT_MUTUAL_AUTH=N
# IF CONNECT_VIA_RELAY
INSTALL_AGENT_RELAY_HTTP_PORT=20080
# ELSE
# INSTALL_AGENT_RELAY_HTTP_PORT=

INSTALL_AGENT_NAME=anthill3-agent
INSTALL_AGENT_DIR=/opt/anthill3/batch/agent

INSTALL_SERVICE=N
INSTALL_SERVICE_NAME=ah3agent
```

```

INSTALL_SERVICE_LOGIN=.\localsystem
INSTALL_SERVICE_PASSWORD=nopass
INSTALL_SERVICE_AUTOSTART=N

agent_count=10

#####
# The installation script.
#####
PREVIOUS_DIR=$(pwd)
PREVIOUS_ANT_HOME=$ANT_HOME
PREVIOUS_CLASSPATH=$CLASSPATH
OUR_ANT_VERSION=1.7.1

SHELL_NAME=$0
SHELL_PATH=$(dirname ${SHELL_NAME})

if [ "." = "$SHELL_PATH" ]
then
    SHELL_PATH=$(pwd)
fi
cd ${SHELL_PATH}

export ANT_HOME=opt/apache-ant-${OUR_ANT_VERSION}

chmod +x opt/apache-ant-${OUR_ANT_VERSION}/bin/ant

export CLASSPATH=

# Run the installation.
i=0
while [ $i -lt "$agent_count" ]
do
    opt/apache-ant-${OUR_ANT_VERSION}/bin/ant \
        "-Dinstall-agent=true" \
        "-Dinstall-server=false" \
        "-Dinstall.java.home=$JAVA_HOME" \
        "-Dinstall.agent.connect_via_relay=$CONNECT_VIA_RELAY" \
        "-Dinstall.agent.jms.remote.host=$INSTALL_AGENT_REMOTE_HOST" \
        "-Dinstall.agent.jms.remote.port=$INSTALL_AGENT_REMOTE_PORT" \
        "-Dinstall.agent.relay.http.port=$INSTALL_AGENT_RELAY_HTTP_PORT" \
        "-Dinstall.agent.remote.port.ssl=$INSTALL_AGENT_REMOTE_PORT_SSL" \
        "-Dinstall.agent.remote.port.mutual_auth=$INSTALL_AGENT_REMOTE_PORT_MUTUAL_AUTH" \
        "-Dinstall.agent.name=$INSTALL_AGENT_NAME-$i" \
        "-Dinstall.agent.dir=$INSTALL_AGENT_DIR-$i" \
        "-Dinstall.service=$INSTALL_SERVICE" \
        "-Dinstall.service.name=$INSTALL_SERVICE_NAME" \
        "-Dinstall.service.login=$INSTALL_SERVICE_LOGIN" \
        "-Dinstall.service.password=$INSTALL_SERVICE_PASSWORD" \
        "-Dinstall.service.autostart=$INSTALL_SERVICE_AUTOSTART" \
        -f install.with.groovy.xml install-non-interactive
    i=$((i+1))
done

# Restore previous state
cd ${PREVIOUS_DIR}
export ANT_HOME=${PREVIOUS_ANT_HOME}
export CLASSPATH=${PREVIOUS_CLASSPATH}

```

Run with Apache Server

The embedded Tomcat instance can be configured to run behind Apache HTTPD. This is desirable to make the Ant-

hillPro UI available on the standard HTTP/HTTPS ports (80/443) without the security risk of running the JVM as root.

- `$ANTHILL3_HOME` should be replaced with your AnthillPro server home directory. Host names (anthill3.example.com) should be replaced by values appropriate to your site.

See also [Optimizing Server Performance](#).

Tomcat Configuration

Locate the Tomcat server configuration file, `server.xml`, which is in `$ANTHILL3_HOME/opt/tomcat/conf`. Add a connector child element to the service element (this should be near the top of the file):

```
<Connector port="8009" minProcessors="5" maxProcessors="256"
protocol="AJP/1.3" />
```

AnthillPro must be restarted for this to take effect.

Apache 2.0 Configuration

Configuring Apache to use `mod_jk` is somewhat complicated. First, for Apache 2.0, use `mod_jk` version 1.2.19. You may need to look in the Tomcat project archives to find the appropriate binary. Later versions only support Apache 2.2. After downloading, extract `mod_jk.so` and add it to the Apache modules directory.

Add the following to Apache's `httpd.conf`:

```
LoadModule jk_module modules/mod_jk.so

JkWorkersFile "conf/workers.properties"

<VirtualHost *:80>
    ServerName anthill3.example.com
    JkMount /* anthill3
</VirtualHost>
```

Create a file called `workers.properties` in the Apache `conf` directory:

```
worker.list=anthill3
worker.anthill3.type=ajp13
worker.anthill3.host=anthill3.example.com
worker.anthill3.port=8009
```

The `worker.anthill3.host` should match the value of `ServerName` in `httpd.conf` and `worker.anthill3.port` must match the value of the `Connector port` in `server.xml`.

Apache 2.2 Configuration

For Apache 2.2, use `mod_jk` version 1.2.21 or later. The binaries are accessible via the normal Tomcat/Jk connect-or download [<http://tomcat.apache.org/connectors-doc/>] page. The configuration is identical to Apache 2.0.

In addition to `mod_jk`, Apache 2.2 adds a new proxy module, `mod_proxy_ajp` designed to work with Tomcat. This module is part of Apache 2.2, so a separate download is not necessary.

The Tomcat configuration is the same as above.

Add these lines to `httpd.conf`:

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so

<VirtualHost *:80>
  ServerName anthill3.example.com
  ProxyPass / ajp://internal.example.com:8009/
  ProxyPassReverse / ajp://internal.example.com:8009/
</VirtualHost>
```

Changing the Context Path on Apache-based Installations

The above directions assume the Anthill UI is running as the root context (that is, the UI would be viewable at `http://anthill3.example.com/`), but it is possible to run the UI with a different context path (e.g., `http://www.example.com/anthill3`). *This is not a fully supported configuration and upgrading may become difficult.*

To change the context path from `/` to `/anthill` on Apache-based installations, do the following: Edit `server.xml` and change the context element to have `path="/anthill"` and `docBase="anthill"`. Then, rename `$ANTHILL3_HOME/opt/tomcat/webapps/ROOT` to `$ANTHILL3_HOME/opt/tomcat/webapps/anthill`.

In `httpd.conf`, update the URLs in the `JkMount` and `ProxyPass . . .` directives:

```
JkMount /anthill/* anthill3
```

or:

```
ProxyPass /anthill ajp://internal.example.com:8009/anthill
ProxyPassReverse /anthill ajp://internal.example.com:8009/anthill
```

Additionally, add `JkMount` and `ProxyPass . . .` lines for the AnthillPro Help:

```
JkMount /anthillpro-help/* anthill3
```

or:

```
ProxyPass /anthillpro-help ajp://internal.example.com:8009/anthillpro-help
```

```
ProxyPass /anthillpro-help ajp://internal.example.com:8009/anthillpro-help
ProxyPassReverse /anthillpro-help ajp://internal.example.com:8009/anthillpro-help
```

Changing the Context Path on Non-Apache Installations

To change the context path if AnthillPro is not behind an Apache server:

1. Shut down AnthillPro.
2. Rename the `opt/tomcat/webapps/ROOT` directory to the context you want to use.
3. Edit `opt/tomcat/conf/server.xml` and change the context path and docbase attribute, for example

```
Context path="/anthill" docBase="anthill"
```
4. Delete the `opt/tomcat/work/Catalina` directory.
5. Start AnthillPro.

Installing Distributed Servers

Installation of Distributed Servers is similar to installing the central AnthillPro server. You will need a Distributed Web Interface license (in addition to the AnthillPro Server license) and, if connecting agents via the Agent Relay, an Agent Relay license.

To complete installation of Distributed servers, do the following:

1. Install the **AnthillPro main server**. See Installing AnthillPro.
2. Retrieve Distributed Servers license and load it in the AnthillPro server UI. See Upload Distributed Servers License.
3. Install the **Distributed Web Interface**. See either Install Distributed Web Interface (Windows) or Install Distributed Web Interface (Linux/Unix).
4. Install the **Agent Relay (optional)**. See either Install Agent Relay (Windows) or Install Agent Relay (Linux/Unix).
5. Install **Agents**. See either Install the Agent with Relay (Windows) or Install the Agent with Relay (Linux/Unix).
6. Configure Agents to use the Distributed Web Interface for Codestation caching (optional). See Distributed Servers and Agent Configuration.

See also Distributed Servers: Best Practices for Installation.

Prerequisites: Distributed Web Interface Installation

- JRE v1.5.0 (or greater) installed in the OS. Set the `JAVA_HOME` environment variable to point to the Java directory.
- The AnthillPro server must already be installed. See Installing AnthillPro.
- You must have a license for the Distributed Web Interface and/or Agent Relay, depending on how you plan to install Distributed Servers. Contact support [<https://support.urbancode.com/>] for more information. See Upload Distributed Servers License.
- The 'admin' account and password, created during AnthillPro server installation, must be available to complete the installation.
- Familiarity with the Distributed Servers system architecture.

Upload Distributed Servers License

Once the AnthillPro server is installed and running (see Installing AnthillPro), retrieve the Distributed Servers license and upload it in the AnthillPro server UI.

1. Go to the AnthillPro download page [<https://support.urbancode.com/>] and log in to your account.
2. Get a copy of your **Distributed Servers** license.
3. Log in to the AnthillPro server as the 'admin'.
4. Go to **System > License** under the Server menu.
5. Paste the license in the **Upload New License** text field.
6. Click **Upload** then **Done**.
7. See Install Distributed Web Interface (Windows) or Install Distributed Web Interface (Linux/Unix).

Install Distributed Web Interface (Windows)

Make sure the AnthillPro server is already installed. See Windows Server Installation from zip File.

1. Download the **anthill3-distributed-web-<version>.zip** file from the Urbancode Support Portal.
2. Expand the zip file using a tool like WinZip. Expanding will create a **anthill3-distributed-web** directory.
3. Open the **anthill3-distributed-web** directory in Windows.
4. Run: **install.cmd**.
5. Provide the following:
 - **Directory where the Distributed Web Interface is to be installed.** Enter the directory where the Distributed Web interface should be installed. Default is **C:\Program Files\anthill3\dist-web**. If the directory does not already exist, enter **Y** to have the installer create it.
 - **Java home.** Enter the Java home used to run the Distributed Web interface. The installer will attempt to find it, or use the **JAVA_HOME** variable. If not found, enter it here.
 - **AnthillPro server IP or host name.** Enter the IP or host name of the AnthillPro server. This is the host, or IP, on which the Distributed Web Interface can contact the main AnthillPro server. To determine the AnthillPro server IP or host name, go to: **AnthillPro UI > System > Server Settings**.
 - **HTTP or HTTPS for AnthillPro server.** If the AnthillPro server is using HTTPS, enter **Y**. Otherwise, enter **N**. Default is **N**. To find if the AnthillPro server uses HTTPS, go to the AnthillPro server's **\conf\server** directory and open the **install.properties** file in a text editor. Check the **install.server.web.always.secure=** property.
 - **AnthillPro server port for web connection.** Enter the port on which the AnthillPro server listens for web connections.
 - If using HTTP, the default port used for web connections is **8080**.
 - If using HTTPS, the default port used for web connections is **8443**.

To find the port, go to the AnthillPro server's `\conf\server\` directory and open the `install.properties` file in a text editor. Check the `install.server.web.port=` property.

- **Legacy communication port.** Enter the legacy port on which the AnthillPro server listens for remote connections. If AnthillPro is being installed for the first time, use the default **4567**.

For 3.6.x communication has been changed, so the legacy port must be updated. To find the port, go to the AnthillPro server's `\conf\server\` directory and open the `install.properties` file in a text editor. Check the `install.server.port=` property.

- **New AnthillPro internal communication port.** Enter the new communication port which the AnthillPro server listens on. The new default value is **7915**.
- **Distributed Web Interface communication port.** Enter the port on which the Distributed Web interface should use for communication. Default is **7917**.
- **Secure communication.** Determine if using secure communication. To use secure connection between the agent, the Relay (if used), and the AnthillPro server, enter **Y**. Default is **N**.
 - **Mutual authentication.** If using secure communication, determine if mutual authentication is required. To require mutual authentication, enter **Y**. Default is **N**.
- **Distributed Web Interface user name.** Enter the user name to use when connecting the Distributed Web interface to the AnthillPro server. Default is **admin**. This is the **admin** account created during installation of the AnthillPro server. The default user name, **admin**, should be used, along with the current password associated with this account.
- **Password.** Enter the password to use when connecting the Distributed Web interface to the AnthillPro server. Default is **admin**. This is the password associated with the **admin** account created during installation of the AnthillPro server. The default user name, **admin**, should be used, along with the current password associated with this account.
- **Name.** Give a name for this Distributed Web node. Each node should have a unique, meaningful name. Default is **dist-web**.
- **Distributed Web IP or host name.** Enter the IP or hostname the Distributed Web will listen on. Default is **0.0.0.0**. The default value will allow the Distributed Web to listen on all IPs associated with the AnthillPro server.
- **HTTP or HTTPS for Distributed Web Interface.** Determine if the Distributed Web should use HTTPS or not. Enter **Y** to use HTTPS, or **N** to use HTTP. Default is **N**.
 - **Distributed Web port for web connections.** If using HTTPS, enter the port on which the Distributed Web will listen for web connections. Default is **8443**.
- **Distributed Web port for web connections.** Enter the port on which the Distributed Web interface will listen for web connections. Default is **8080**.

6. Start the Distributed Web Interface.

- To start the Distributed Web in a new shell, go to the Distributed Web's `\bin` directory and run `start_ah3web.cmd`.

Or:

- To start the Distributed Web from the command line, run `ah3web run`.

7. If using the Agent Relay, see Install Agent Relay (Windows). Otherwise, follow the instructions given in Installing the Agent (Windows).

Install Agent Relay (Windows)

If Agent communication is routed through the relay proxy, the Relay must be installed before installing the agents that will use it.

Before installing the Agent Relay, make sure the AnthillPro server is installed (see Windows Server Installation from zip File) and that the license has been uploaded to the AnthillPro server (see Upload Distributed Servers License).

1. Download the **anthill3-agent-relay-<version>.zip** file.
2. Expand the zip file using a tool like WinZip. Expanding will create a **anthill3-agent-relay** directory.
3. Open the **anthill3-agent-relay** directory.
4. Run the install script **install.cmd**.
5. Provide the following:
 - **Location of Agent Relay.** Give the directory where the Agent Relay should be installed. Default is **C:\Program Files\anthill3\relay**. If the directory does not already exist, enter **Y** to have the installer create it. Default is **Y**.
 - **Java home.** Enter the Java home used to run the Distributed Web interface. The installer will attempt to find it, or use the **JAVA_HOME** variable. If not found, you will need to enter it here.
 - **Name.** Give a name for the relay. Each Agent Relay should be given a unique, meaningful name. Default name is **agent-relay**.
 - **Agent Relay IP or host name.** Enter the IP or hostname the Agent Relay will listen on. Default is **0.0.0.0**. The default value will allow the Agent Relay to listen on all IPs associated with the AnthillPro server.
 - **Port Relay listens on for HTTP requests.** Enter the port on which the Agent Relay should listen for HTTP requests coming from the agent(s). Default is **20080**.
 - **Port Relay uses for communication.** Enter the port on which the Agent Relay will use for communication with the agent. Default is **7916**.
 - **AnthillPro server IP or host name.** Enter the IP or host name of the AnthillPro server. This is the host, or IP, on which the Agent Relay can contact the main AnthillPro server. To determine the AnthillPro server IP or host name, go to: **AnthillPro UI > System > Server Settings**.
 - **AnthillPro server communication port.** Enter the port which the AnthillPro server uses for communication. If you used the default value during server installation, use **7915**. If the AnthillPro server uses a different port, go to the AnthillPro server's **\conf\server** directory and open the **install.properties** file in a text editor. The port is listed in the **install.server.jms.port=** property.
 - **Secure communication.** Determine if using secure communication between the server, the relay, and the agents. To use secure connection, enter **Y**. Default is **N**.
 - **Mutual authentication.** If using secure communication, determine if mutual authentication is required. To require mutual authentication, enter **Y**. Default is **N**.
6. **Optionally**, install Agent Relay as Windows Service:

For newer versions of Windows, you will need to run as Administrator from the command line to install as a Windows Service. For example: `C:\Windows\system32>`.

- Open the command line and point it to the Agent Relay's `\bin\service` directory.
- Run: **ah3relay.cmd install**.
- Follow the instructions on the installer to run the service with **Log on Service** rights.
- Enter the user account name, including the domain path, to run the service. Default is `.\localsystem`.
- Enter **Y** to start service automatically. Default is **N**. Service must be manually started the first time.
- To remove the Agent Relay as Windows Service, run: **ah3relay.cmd remove**.

7. Start the Agent Relay.

- To start the Agent Relay in a new shell, go to the Agent Relay's `\bin` directory and run **start_ah3relay.cmd**.

Or:

- To start the Agent Relay from the command line, run **ah3relay run**.

8. See Install the Agent with Relay (Windows).

Install the Agent with Relay (Windows)

Make sure the AnthillPro server, Distributed Web Interface (if used), and Agent Relay are installed. See Windows Server Installation from zip File; Install Distributed Web Interface (Windows); and Install Agent Relay (Windows).

If you are not using the Agent Relay, follow the agent installation instructions found in Installing AnthillPro.

1. Download the **anthill3-<version>.zip** file. If you only installing agents, you can download the **anthill3-agent-<version>.zip** file, which is much smaller.
2. Expand the zip file using a tool like WinZip. Expanding will create a **anthill3-install** directory.
3. Open the **anthill3-install** directory.
4. Run the install script **install-agent.bat**.
5. Provide the following information:
 - **Directory** where the AnthillPro **agent** should be installed. For example: `C:\Program Files\anthill3\agent`. To have the installer create the directory, enter **Y**.
 - **Java home**. Enter the Java home used to run the Agent. The installer will attempt to find it, or use the `JAVA_HOME` variable. If not found, you will need to enter it here.
 - **Relay connection**. Enter **Y**. Before continuing, make sure the Agent Relay is already installed (separate download). See Install Agent Relay (Windows).
 - **Host name** or **address** of the Agent Relay. Default is **localhost**.
 - **Port** which the Agent Relay will use for communication. Default is **7916**.

- Determine if communication should be secured using **SSL**. Default is **N**.

*When using SSL, you need to turn it on for both the **server and the agent** -- otherwise the agents will not be able to connect to the server. If you are using the agent relay, ensure that **SSL is turned on for all three components: server, agent, and agent relay**. This rule also applies if using mutual authentication.*

- If using SSL, determine if **mutual authentication** should be used. Default is **N**.
- **HTTP port** the Agent Relay will use for web requests. Default is **20080**.

6. **Name**. Give a unique, meaningful name for every agent installed.

7. Type **Y** to automatically install the AnthillPro agent as a Windows service. Type **N** to complete installation process.

For newer versions of Windows, you will need to run as Administrator from the command line to install as a Windows Service. For example: `C:\Windows\system32>`.

8. If automatically installing the AnthillPro agent as a Windows service, enter:

- Service **unique name**. Default is **ah3agent**. (A unique name is required for each instance.)
- **Login** for installed agent service. Default is `.\localsystem`.
- **Yes** to automatically start service.

To manually install the AnthillPro agent as a Windows service:

For newer versions of Windows, you will need to run as Administrator from the command line to install as a Windows Service. For example: `C:\Windows\system32>`.

- Run **ah3agent.cmd install uniqueservicename** (located in `bin\service` folder of the AnthillPro directory).
- Provide the service **name** and **account information**.

To run the AnthillPro agent:

1. In Windows navigate to the **AnthillPro server directory** created during the installation. If you took our suggestion, then this is: `C:\Program Files\anthill3\agent`. Enter the **bin** directory.
2. Run the start script: **start_ah3agent.cmd**.
3. Alternatively, from the **command line** run the start command `bin\ah3agent.cmd start`.
4. See Distributed Servers and Agent Configuration.

Install Distributed Web Interface (Linux/Unix)

Make sure the AnthillPro server is already installed. See Linux/Unix Server Installation from tar.gz File.

1. Download the **anthill3-distributed-web-<version>.tar.gz** file.
2. Expand the archive. Expanding will create the **anthill3-distributed-web** directory.
3. Open the **anthill3-distributed-web** directory.
4. Run: **install.sh**.
5. Provide the following:
 - **Directory where the Distributed Web Interface is to be installed.** Enter the directory where the Distributed Web interface should be installed. Default is **/opt/anthill3/dist-web**. If the directory does not already exist, enter **Y** to have the installer create it.
 - **Java home.** Enter the Java home used to run the Distributed Web interface. The installer will attempt to find it, or use the **JAVA_HOME** variable. If not found, you will need to enter it here.
 - **AnthillPro server IP or host name.** Enter the IP or host name of the AnthillPro server. This is the host, or IP, on which the Distributed Web Interface can contact the main AnthillPro server. To determine the AnthillPro server IP or host name, go to: **AnthillPro UI > System > Server Settings**.
 - **HTTP or HTTPS for AnthillPro server.** If the AnthillPro server is using HTTPS, enter **Y**. Otherwise, enter **N**. Default is **N**. To find if the AnthillPro server uses HTTPS, go to the AnthillPro server's **/conf/server/** directory and open the **install.properties** file in a text editor. Check the **install.server.web.always.secure=** property.
 - **AnthillPro server port for web connection.** Enter the port on which the AnthillPro server listens for web connections.
 - If using HTTP, the default port used for web connections is **8080**.
 - If using HTTPS, the default port used for web connections is **8443**.To find the port, go to the AnthillPro server's **/conf/server/** directory and open the **install.properties** file in a text editor. Check the **install.server.web.port=** property.
 - **Legacy communication port.** Enter the legacy port on which the AnthillPro server listens for remote connections. If AnthillPro is being installed for the first time, use the default **4567**.

For 3.6.x communication has been changed, so the legacy port must be updated. To find the port, go to the AnthillPro server's **/conf/server/** directory and open the **install.properties** file in a text editor. Check the **install.server.port=** property.
 - **New AnthillPro internal communication port.** Enter the new communication port which the AnthillPro server listens on. The new default value is **7915**.
 - **Distributed Web Interface communication port.** Enter the port on which the Distributed Web interface should use for communication. Default is **7917**.
 - **Secure communication.** Determine if using secure communication. To use secure connection between the agent, the Relay (if used), and the AnthillPro server, enter **Y**. Default is **N**.
 - **Mutual authentication.** If using secure communication, determine if mutual authentication is required. To require mutual authentication, enter **Y**. Default is **N**.
 - **Distributed Web Interface user name.** Enter the user name to use when connecting the Distributed Web interface to the AnthillPro server. Default is **admin**. This is the **admin** account created during installation of the AnthillPro server. The default user name, **admin**, should be used, along with the current password associated with this account.

- **Password.** Enter the password to use when connecting the Distributed Web interface to the AnthillPro server. Default is **admin**. This is the password associated with the **admin** account created during installation of the AnthillPro server. The default user name, **admin**, should be used, along with the current password associated with this account.
 - **Name.** Give a name for this Distributed Web node. Each node should have a unique, meaningful name. Default is **dist-web**.
 - **Distributed Web IP or host name.** Enter the IP or hostname the Distributed Web will listen on. Default is **0.0.0.0**. The default value will allow the Distributed Web to listen on all IPs associated with the AnthillPro server.
 - **HTTP or HTTPS for Distributed Web Interface.** Determine if the Distributed Web should use HTTPS or not. Enter **Y** to use HTTPS, or **N** to use HTTP. Default is **N**.
 - **Distributed Web port for web connections.** If using HTTPS, enter the port on which the Distributed Web will listen for web connections. Default is **8443**.
 - **Distributed Web port for web connections.** Enter the port on which the Distributed Web interface will listen for web connections. Default is **8080**.
6. Start the Distributed Web Interface.
- Open the Distributed Web interface's '/bin' directory.
 - Run: 'ah3web run'.
- OR:
- Start the Distributed Web interface in the background. Run: 'ah3web start'.
7. Retrieve the Distributed Web Interface license.
8. Go to the AnthillPro server UI. Navigate to **System > Licenses** and paste the license in the **Upload New License** text field. Click **Upload** then **Done**.
9. See Install Agent Relay (Linux/Unix) and Managing Distributed Servers.

Install Agent Relay (Linux/Unix)

If Agent communication is routed through the relay proxy, the Relay must be installed before installing the agents that will use it.

Before installing the Agent Relay, make sure the AnthillPro server is installed (see Linux/Unix Server Installation from tar.gz File) and that the license has been uploaded to the AnthillPro server (see Upload Distributed Servers License).

1. Download the **anthill3-agent-relay-<version>.tar.gz** file.
2. Expand the file to create the **anthill3-agent-relay** directory.
3. Open the **anthill3-agent-relay** directory.
4. Run the install script **install.sh**.
5. Provide the following:

- **Location of Agent Relay.** Give the directory where the Agent Relay should be installed. Default is `/opt/anthill3/relay`. If the directory does not already exist, enter **Y** to have the installer create it. Default is **Y**.
 - **Java home.** Enter the Java home used to run the Distributed Web interface. The installer will attempt to find it, or use the `JAVA_HOME` variable. If not found, you will need to enter it here.
 - **Name.** Give a name for the relay. Each Agent Relay should be given a unique, meaningful name. Default name is **agent-relay**.
 - **Agent Relay IP or host name.** Enter the IP or hostname the Agent Relay will listen on. Default is **0.0.0.0**. The default value will allow the Agent Relay to listen on all IPs associated with the AnthillPro server.
 - **Port Relay listens on for HTTP requests.** Enter the port on which the Agent Relay should listen for HTTP requests coming from the agent(s). Default is **20080**.
 - **Port Relay uses for communication.** Enter the port on which the Agent Relay will use for communication with the agent. Default is **7916**.
 - **AnthillPro server IP or host name.** Enter the IP or host name of the AnthillPro server. This is the host, or IP, on which the Agent Relay can contact the main AnthillPro server. To determine the AnthillPro server IP or host name, go to: **AnthillPro UI > System > Server Settings**.
 - **AnthillPro server communication port.** Enter the port which the AnthillPro server uses for communication. If you used the default value during server installation, use **7915**. If the AnthillPro server uses a different port, go to the AnthillPro server's `/conf/server/` directory and open the **install.properties** file in a text editor. The port is listed in the **install.server.jms.port=** property.
 - **Secure communication.** Determine if using secure communication between the server, the relay, and the agents. To use secure connection, enter **Y**. Default is **N**.
 - **Mutual authentication.** If using secure communication, determine if mutual authentication is required. To require mutual authentication, enter **Y**. Default is **N**.
6. Start the Agent Relay.
- To start the Agent Relay in a new shell, go to the Agent Relay's `/bin` directory and run **ah3relay start**.
- Or:
- To start the Agent Relay from the command line, run **ah3relay run**.
7. See Install the Agent with Relay (Linux/Unix).

Install the Agent with Relay (Linux/Unix)

Make sure the AnthillPro server and Distributed Web Interface are installed. See Linux/Unix Server Installation from tar.gz File and Install Distributed Web Interface (Linux/Unix).

If you are not using the Agent Relay, follow the agent installation instructions found in Installing AnthillPro.

1. Download the **anthill3-<version>.tar.gz** file. If you only installing agents, you can download the **anthill3-agent-<version>.tar.gz** file, which is much smaller.
2. Open a **UNIX shell** to the directory containing the above downloaded file.
3. Extract the downloaded file. Type: **tar -zxf anthill3-<version>.tar.gz**.

- On some installations of Solaris and HP-UX the default tar command will not properly handle our tar files. You may need to use `\ install GNU tar`.

When installing AnthillPro on Solaris, it is recommended to use korn shell (ksh).

4. cd **anthill3-install**.

5. Run the install script: **./install-agent.sh**.

6. Provide the following information:

- **Directory** where the AnthillPro **agent** should be installed. For example: **/opt/anthill3/agent** . To have the installer create the directory, enter **Y**.
- **Java home**. Enter the Java home used to run the Agent. The installer will attempt to find it, or use the `JAVA_HOME` variable. If not found, you will need to enter it here.
- **Relay connection**. Enter **Y**. Before continuing, make sure the Agent Relay is already installed (separate download). See Install Agent Relay (Linux/Unix).
- **Host name** or **address** of the Agent Relay. Default is **localhost**.
- **Port** which the Agent Relay will use for communication. Default is **7916**.
- Determine if communication should be secured using **SSL**. Default is **N**.

*When using SSL, you need to turn it on for both the **server and the agent** -- otherwise the agents will not be able to connect to the server. If you are using the agent relay, ensure that **SSL is turned on for all three components: server, agent, and agent relay**. This rule also applies if using mutual authentication.*

- If using SSL, determine if **mutual authentication** should be used. Default is **N**.
- **HTTP port** the Agent Relay will use for web requests. Default is **20080**.

7. Enter name of **agent**.

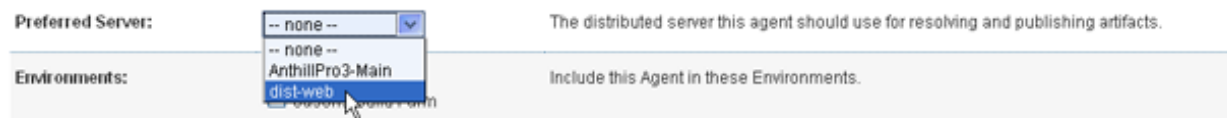
To run the AnthillPro agent, follow the steps below:

1. Open a **UNIX shell** to the directory where you installed the agent. If you took our suggestion, then this is: **/opt/anthill3/agent**.
2. cd **bin**.
3. Run the script: **./ah3agent start**.

See Distributed Servers and Agent Configuration.

Distributed Servers and Agent Configuration

Configure the Agent to use the Distributed Web Interface for Codestation caching. Follow the instructions given in Configure Agent(s). From the **Preferred Server** drop-down menu, select the Distributed Server you want this agent to receive work from.



Agents that are installed on the same LAN as the distributed server will maximize server performance. See Managing Distributed Servers.

Using SSL with Distributed Web Interface

It is possible to configure the AnthillPro server to use SSL with the Distributed Web Interface. To do so, you will need to import an existing certificate and switch to https. Before you begin, make sure AnthillPro is not running any jobs (you can do this on the Current Activity page). The ports listed in the examples may be different than those used by your AnthillPro system.

1. **Shutdown** the AnthillPro server and backup the following files:

- `conf/server/installed.properties`
- `opt/tomcat/conf/server.xml`
- `opt/tomcat/webapps/ROOT/WEB-INF/web.xml`

2. **Edit** `conf/server/installed.properties` file:

- **Change:** `install.server.web.always.secure=Y`

Add: `install.server.web.https.port=8443`

3. **Edit** `opt/tomcat/conf/server.xml` file:

- **Change:** port 8080 connector

Add: `redirectPort="8443"`

Add a new connector:

```
<Connector port="8443"
  address="0.0.0.0"
  maxThreads="150"
  minSpareThreads="25"
  enableLookups="false"
  acceptCount="100"
  debug="0"
  connectionTimeout="20000"
  disableUploadTimeout="true"
  algorithm="SunX509"
  scheme="https"
  secure="true"
  clientAuth="false"
  sslProtocol="TLS"
  keystoreFile="conf/tomcat.keystore"
  keystorePass="changeit" />
```

4. **Edit** `opt/tomcat/webapps/ROOT/WEB-INF/web.xml` file:

- **Replace:** `<!-- security-constraints -->` with:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>
      Tools
    </web-resource-name>
    <url-pattern>/tools/*</url-pattern>
  </web-resource-collection>
</security-constraint>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>
      Automatic SSL Forwarding
    </web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>
      CONFIDENTIAL
    </transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

5. To **import certificate** go to `opt/tomcat/conf` directory and run the following command (assuming Java / bin directory is on your path):

- `keytool -import -noprompt -alias tomcat -file path_to_certificate_file.crt -storepass "changeit" -keystore tomcat.keystore`

6. **Start** server.

Chapter 9. Setting Up the Server

Before you can start buildign projects, it is recommended to take some time and set up the AnthillPro server's global settings. At the minimum, most users find the following necessary:

1. License activation.If not already done so, obtain a license and install it in the server. AnthillPro will not run without a license.
2. Set up security and users. If you are going to have more than the "admin" user, you will most likely need to configure security and add users.
3. Assign agent to an environment (configure agent). All builds and other similar processes run on an agent. In order for an agent to be available for a build, it needs to be assigned to an environment.
4. Set up the global repository. You build projects need to be associated with a specific SCM. You will need to set the global SCM integration prior to project configuration.

Installing a License

As part of installation, you should have already download and activated your license. If not, you will need to go to Supportal [<http://www.support.urbancode.com/>], the Urbancode support portal, where you downloaded the installation package and retrieve the license:

1. Go to Supportal [<http://www.support.urbancode.com/>] to retrieve the license from your account.
2. Go to **TEAMS/USERS > Licenses**. If you do not see a license, either one does not exist for your account or you do not have permissions to download a license. Please contact your sales representative for more information.
3. Select the **view license** link on the right hand side of the appropriate license.
4. In the pop-up, click **download**.
5. Open the license, copy it, and paste it in the **License** field as outlined in either the Windows Installation or Linux/Unix Installation section.

Basic Security and User Configuration

In AnthillPro, you have detailed control over what a user can see and do. The system enables you to map your organizational structure by teams, activities, etc. For example, you can set up AnthillPro so that a developer only sees the projects they work on, or the QA team can only access the build artifacts.

Security management begins with Roles. In turn, each Role has corresponding Permissions to either restrict or allow a user to perform tasks, view pages, etc. Once the Roles and Permissions have been configured, Authorization Realms realms and then Authentication Realms are configured. Once security is configured, Audits may be performed for the who-when changes Administrative users make to the system.

While you can configure a project as the admin user, and allow multiple people to concurrently log on to AnthillPro as the same user, it is advisable to add users -- and assign them a default role -- before going forward. This will allow you to see how AnthillPro integrates with other tools such as LDAP as well as allow you to control who gets notified for which events.

To set up security and add users, see the Security section.

Configure Agent

When an agent is installed and started on a host (i.e., a different machine), it contacts the AnthillPro central server. When a connection is established, the agent will show up under the Available tab (go to **Agents > Agent > Available**). The agents listed on the Available tab must be configured before they can start accepting work from the server and running commands, such as invoking your build script.

To configure an agent:

1. Go to **Agents > Agent**.
2. Select the **Available** tab. This page lists all the agents that can access the server, but are not available to run a build. If you installed an agent but it does not appear on this tab, make sure the agent is running and able to connect to the server (e.g., there is no firewall blocking server-agent communication).
3. Provide the following:
 - **Name.** Give the agent a descriptive name. The current name of the agent was given during the installation process. AnthillPro will automatically update the agent if its name is changed.
 - **Throughput Metric.** The throughput metric provides a hint to AnthillPro's load balancer. In a busy Build-Farm, it may be that several machines could handle a request, but each is already running builds. To determine which machine is best equipped to handle the additional build, the load balancer compares the machine's throughput metric number to the number of jobs that are running on it. So a server with a metric of 10 will get a third job assigned to it before a server with a metric of 1 gets its second job.
 - **Maximum Job Count.** Agents can also be given a maximum number of jobs that they may run. If all agents a job can run on are at their maximum, it waits queued until a machine frees up.
 - **Preferred Server.** If utilizing distributed servers, select the appropriate server. In most cases, the agent should be connected to the preferred server via a LAN. Otherwise, select "none". See Distributed Servers and Agent Configuration.
 - **Environments.** You must check at least one environment. For running builds, most people select Build-Farm. This information will be used during project configuration.

If you are setting up AnthillPro for the first time, you can configure the agent to participate in all of the default environments (Build-Farm, Production, and QA). See Environment Management for a more detailed discussion.
4. Click **Set** then **Done**. The agent will automatically move to the **Configured** tab.

See also Agent Configuration for a more detailed discussion.

Agent Properties (Installation)

The Properties tab (see below) on the agent configuration page allows you to view or set custom variables on the agent.

- The **custom variables** can indicate where build or testing tools are installed.
- In the **Locked Variables** section, review the system and environment variables. Those are often used in agent fil-

ters.

See also Agent Configuration for a more detailed discussion.

Agent Security tab

Manage user access on the **Security** tab. Administrators can define what roles have access to read, write, or determine security for agents.

See also Agent Configuration for a more detailed discussion.

Using Agent Proxies

Once the agent has been installed, a proxy may be set up. Use an agent proxy for any agents where the direct agent-server communication is prohibited. Once enabled, the proxy allows the agent to send logs, reports and other artifacts to the server. See Using Agent Proxies.

Set Up Repositories

The SCM integrations enable AnthillPro to check out code, access the changelog, and label the repository (where supported). To do this, AnthillPro is first configured with your repository type at the System level, and then each workflow is associated with the source to be built.

The SCM integrations are implemented as job steps for a build job. Once you have completed source configuration, you can use the Job Wizard to automatically add steps to your build job -- this ensures that AnthillPro will consistently checkout and build the correct code.

Each SCM integration typically performs the following for any repository:

- **Checkout.** This step enables you to define which version of code to check out from the SCM. For example, you can configure this step to checkout the latest source code; or perform a checkout based on a branch, label, date, etc. (depending on what your SCM supports).
- **Get-changelog.** The retrieved changelog is usually based on the changes made since the previous build. This step enables AnthillPro to extract data from the SCM and then store it in the AnthillPro data warehouse. Since AnthillPro stores the changelogs, it can parse the data, allowing you to override the default behavior. For example, you can select a starting point for the changelog based on criteria such as the latest production build.
- **Label.** AnthillPro can also apply a label to the source code used in the build (e.g., snapshot, baseline). This unique identifier for a build can be used to recreate a build if necessary.
- **SCM-specific commands.** For most repository types, AnthillPro can also perform tasks only supported by that particular SCM.

To set up the global repository configuration, choose from the following (it is possible to configure multiple global repositories):

AccuRev	Perforce
ClearCase	PVCS
CVS	Rational Team Concert (SCM)
Dimensions	StarTeam
File System	Subversion (SVN)

Git	Synergy
Harvest	TFS Source Control
(Integrity) MKS Source Integrity	Vault (SourceGear)
Mercurial	Visual SourceSafe (VSS)

Using Commit and Other Triggers

Enabled on the workflow, a trigger is an automated mechanism for kicking off a process, such as an automated build. If you use the repository-type trigger, AnthillPro will kick off a build every time a change is detected. In AntihillPro, you can use one of three trigger types:

- **Scheduled trigger.** The simplest type to configure, the scheduled trigger fires either on a regular interval (e.g., every few minutes, hours, etc.) or can utilize a Cron expression for irregular, but recurring, schedules. When used for CI, the scheduled trigger polls the SCM for source changes. If changes are found, a build is kicked off. See Use Agent Filters and Quiet Periods for more detailed information.
- **Repository (commit) trigger.** Many SCM types support commit triggers that allow AnthillPro to kick off a build at the time source changes are made. Repository triggers are far more efficient than scheduled triggers, as they don't poll the SCM, and are the most common trigger type used for CI builds. See Use Agent Filters and Quiet Periods for more detailed information.

If you are unsure if your SCM supports commit triggers, please check the SCM Tools section for instructions on configuring a commit trigger.

- **Event trigger.** An advanced trigger type, the event trigger allows for a custom filter that taps into the AnthillPro event service and triggers actions when certain events occur. For example, an event trigger would listen for build completed events, check those events against the project's dependencies, and force a build of the project if a dependency builds successfully.

This section is for general trigger usage. If you followed the instructions for setting up a CI build, you should have already configured a trigger. If not, you can follow the process below. Otherwise, you can use this section as a basis for switching to a repository trigger.

To activate a commit trigger, ensure that your SCM type supports it. Generally, AnthillPro provides a hook script for most SCMs that support commit triggers. If you are unsure, please check the SCM Tools section for instructions on configuring a commit trigger. Generally, trigger configuration is as follows:

1. Go to **Administration** and select the workflow of the project you want to trigger.
2. On the **Workflow** page, select the **Triggers** tab and then click the **New Trigger** button.
3. Select **Trigger** type from the drop-down menu and click **Select**.
4. Configure trigger. Generally, you have the following options:
 - **Name.** Give a name that identifies this trigger.
 - **Force.** By default, AnthillPro does not force a build if the trigger does not find any code changes when the trigger fires. It is rare that you will want to force a build every time the trigger fires, so leave this field blank.
 - **Enabled.** By default, when you create a new trigger it is enabled, which is what you want for a CI build.

- Click **Save** then **Done**.

5. The next time the trigger fires, and it detects a source change, the build will take place. It will be identified (on the Dashboard) by the type of trigger that kicked off the build.



Workflow	Status	Stamp	When	Duration	Why	Tests	Issues	Changes
Trunk Build Workflow	Complete	1.0.14	8/18/09 12:10 PM	27 s	Scheduled	0 / 0	0	0 / 0

Note that if you are not seeing builds, it is most likely because no changes have been made to the source code.

Chapter 10. Create a New Project

Creating a Project brings together two other AnthillPro concepts: Lifecycle Models and Environment Groups. When setting up a project for the first time, you can use the default options (given below) for simplicity. However, it is also possible to configure custom Models and Environment Groups: if you go this route, it may be worth reading the Concepts section first.

Setting Up a Build

When getting started with AnthillPro, it is helpful to think of the build process (simply called the "build") as having three stages:

- **Input.** Consists of the source code, located in an SCM. To run a build, AnthillPro will need to know where to find the source code.
- **Transformation.** The actions necessary to generate the output. In AnthillPro, this corresponds to the workflow and job.
- **Output.** The result of the transformation, called the build artifacts.

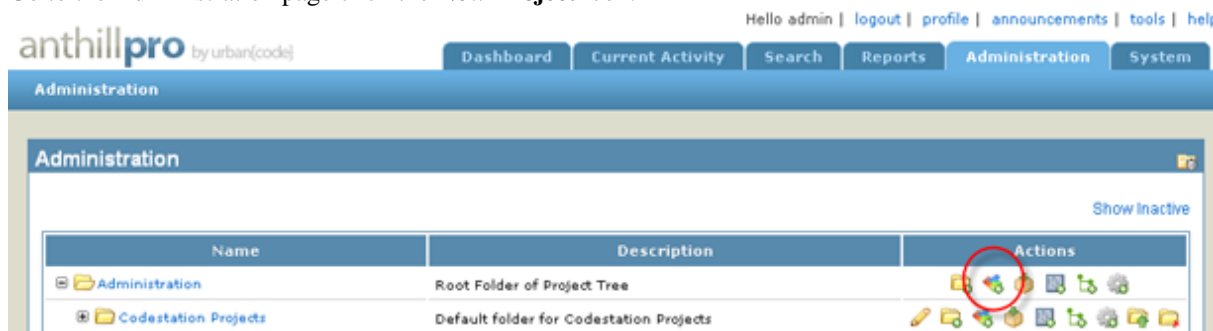
To achieve a successful build:

1. Make sure the AnthillPro server and at least one agent are installed and running.
2. Know the AnthillPro server URL (e.g., <http://localhost:8080/>), as well as your user name and password, to log on to AnthillPro.
3. The URL of the source repository (e.g., CVS, Subversion, TFS, etc.). Because AnthillPro will check out a copy the source code when a build is requested, you will also need to give AnthillPro access to the repository, usually assigning a user account to AnthillPro.
4. **Configure a new project.** This project, called a Life-Cycle Based project in AnthillPro, is composed of at least one workflow that will run the build.
5. **Configure the repository (SCM).** Let AnthillPro know about the source repository before you configure a workflow.
6. **Create a new workflow.** The workflow is AnthillPro's basic unit of process automation. It is where you define how the *transformation* (i.e., the build) will take place, as well where you will perform source configuration.
7. **Create a new job.** The job is the building block of the *transformation*, and is composed of a series of actions the server must perform (called "steps" in AnthillPro). The job, in turn, is added to the workflow.
8. **Complete build process configuration.** Before you can run a build, you need to tie the workflow and job into a complete build process.
9. **Run a build.** Once the configuration is done, you are ready to build!

Configure Project

To create a new project:

1. Go to the Administration page click the **New Project** icon.



2. On the **Create a Project** page, check the **Life-Cycle Based** project and click **Select**. In AnthillPro, it is the Life-Cycle Based project that is used to run a build.

For those just starting out, it is best to use the default settings and configurations, where possible, when setting up your build. Once you become more familiar with AnthillPro, you can begin to use the more advanced features.

- **Name.** Give a name for this project. Typically, the name should reflect the type of project. You can map the name to your existing build infrastructure.
- **Description (optional).** Give a description of this project. The descriptions allow you to convey more information about the project.
- **Life-Cycle Model.** Select the Example Life-Cycle Model. This is the default model that ships with AnthillPro.

If you want to use your own Life-Cycle Model, see Using a Custom Lifecycle Model.

- **Environment Group.** Select the Default Environment Group that ships with AnthillPro.
- Click **Save**.

Using a Custom Lifecycle Model

A Life-Cycle Model allows you to create a reusable template that maps your organizational structure. For example, a typical set-up would be DEV, QA, PROD process (i.e., pipeline or life-cycle): a build starts out in development, is deployed to quality assurance for testing, and then finally sent to production for release. You would configure the Life-Cycle Model to apply a new status when a build is sent to QA, and one when the build is sent to PROD. Likewise, a stamp style (essentially a build identifier) can be applied to each build that corresponds to the status. This enables you to know exactly which build is in which environment because the status and stamp are recorded on the Dashboard.

Life-Cycle Models, then, give you control over how a build is identified, the different stages a build must go through on its way to the end user, how artifacts are handled, and which clean-up policies are enforced.

Life-Cycle Models are closely associated with AnthillPro's Build Life concept: Much of what you configure in the Life-Cycle Model determines how a Build Life is identified and used. See Build Life for more.

Once a Life-Cycle Model is created, it may be used for multiple projects with similar Life-Cycles without having to reconfigure a list of Statuses, Stamps, Artifact Sets, and Cleanup for the new project. This enables you to create system-wide standards that every team must use when configuring projects.

When getting started, it is usually best to use the default Lifecycle Model until you are familiar with how the models work. Once you have few builds going, and even deploying, you can create your own model that better reflects your organizational structure.

See Using Life-Cycle Models for a complete discussion.

Configure the Repository (SCM)

Before you can configure a project's build workflow, you need to let AnthillPro know about the repository type you use (if not already done so as outlined in the Setting Up the Server section). The information you give in this section will eventually be shared by multiple projects.

1. Select your repository type from the list below and follow the instructions.

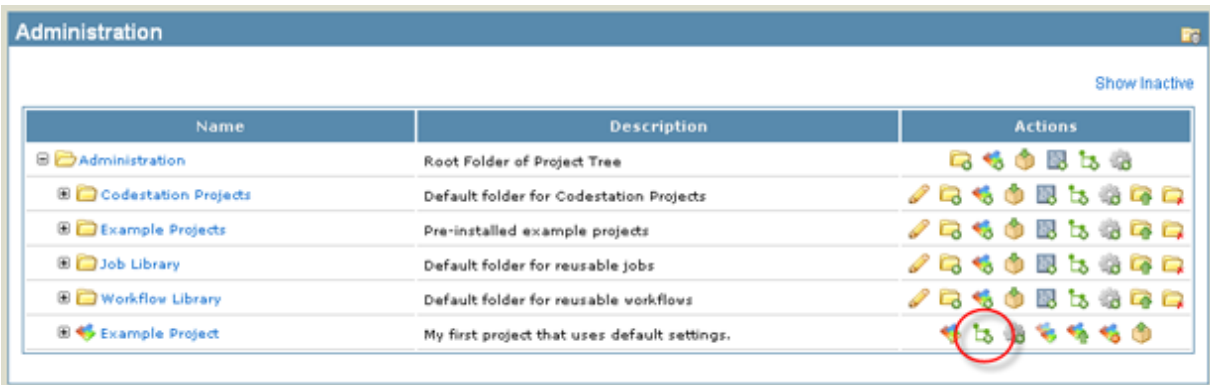
AccuRev	Perforce
ClearCase	PVCS
CVS	Rational Team Concert (SCM)
Dimensions	StarTeam
File System	Subversion (SVN)
Git	Synergy
Harvest	TFS Source Control
(Integrity) MKS Source Integrity	Vault (SourceGear)
Mercurial	Visual SourceSafe (VSS)

2. See Create a New Workflow.

Create a New Workflow

The workflow is AnthillPro's basic unit of process automation for a project. It is where you define how the *transformation* (build) will take place, as well as the source code that corresponds to the project (source configuration). During workflow creation, you will need to provide further information about the SCM type you configured on the System page.

1. Select the **Add Workflow** icon of your new project.



2. Next, check **Originating** workflow and click **Select** on the New Workflow page. In AnthillPro, it is the Originat-

ing workflow that is used to run build. Non-Originating workflows are secondary processes (such as deployments) that can be run on the build.

3. Workflow configuration:

- **Name.** Give a name for this workflow. Typically, the name should reflect the type of work to be performed when this workflow runs. For example, "Trunk Build."
- **Description (optional).** Give a description of this workflow. The description allows you to convey more information about the workflow.
- **Environment.** Assign the workflow to the Build-Farm environment. In AnthillPro, the environment is the location where the build takes place. AnthillPro ships with 3 default environments:
 - **Build-Farm.** Generally, this is the environment that all builds take place within, and is most closely associated with Development. This is the environment you want your builds to run in.
 - **QA.** This environment is provided for performing quality assurance testing. Most often, when you are sending a build to testing, you want that workflow to run in this environment. See *Setting Up a Deployment* for more.
 - **Production.** This environment is typically used to deploy your builds once they are ready for production. See *Setting Up a Deployment* for more.
- **Notification Scheme.** Select the **Default Notification** scheme. By default, AnthillPro will send an e-mail notification to all AnthillPro users on completion of the workflow. For now, don't worry if you aren't getting any e-mails. This will be covered in more detail when you set up a Continuous Integration build.
- **Priority.** Use the default Normal Priority. Once you have a lot of projects going in AnthillPro, you may need to prioritize which workflows run first.
- **Skip Quiet Period.** Select No, the default value. This will ensure that the build takes the most recent source code. It is not recommended to skip the workflow's quiet period under most circumstances. By default, AnthillPro based the quiet period on the changelog, which is what you need when configuring scheduled builds. See *Use Agent Filters and Quiet Periods* for more.
- Click **Save**.

4. Source Configuration. Select the repository you want to associated with this workflow and click **Set**. This should correspond to the repository you configured earlier.

Once the repository is selected, you will be able to configure additional properties that tie this workflow to the specific source that will be built. See the **Source Configuration** section of the repository type used by this workflow. When done, go to the next item.

AccuRev	Mercurial
ClearCase Base Dynamic View	Perforce
ClearCase Base Snapshot View	PVCS
ClearCase UCM Dynamic View	Rational Team Concert (SCM)
ClearCase UCM Snapshot View	StarTeam
CVS	Subversion (SVN)
Dimensions	Synergy
File System	TFS Source Control
Git	Vault (SourceGear)
Harvest	Visual SourceSafe (VSS)

(Integrity) MKS Source Integrity

5. **Set stamp value.** To complete workflow configuration you will need to add a value to the default stamp (DEV) under the Stamping Strategies menu. The stamp enables you to apply a custom identifier to a build, in addition to the Build Life which AnthillPro automatically generates.

*Without a Stamp Style **value**, AnthillPro will not be able to complete a build.*

- Click the **Create Value** icon under the Action menu.



- **Stamp.** Give the value AnthillPro will use to stamp the build. The simplest way to make the stamp unique for each build is to associate it with the Build Life number that AnthillPro automatically generates. For now, use the following to generate a stamp:

```
build_${bsh:BuildLifeLookup.getCurrent().getId() }
```

The short BeanShell script looks up the current build ID and then applies that number to the stamp. So for Build Life 1 the stamp will be **build_1**; for Build Life 2 the stamp will be **build_2**, etc.

For a detailed discussion on Stamps, see the Stamping section.

- Click **Save**.
6. Click **Done**.
 7. See Create a New Job.

Originating (Build) Workflow and Artifacts Sets

Once a build has completed, the generated artifacts are still in the working directory. To have AnthillPro track the artifacts, and make them available for deployments, etc., the artifacts need to be captured and delivered to Codestation, the artifact management system. Once the artifacts are in Codestation, AnthillPro will be able to keep track of them as well as any other process, such as deployment to testing, etc., the artifacts are subjected to.

Capture and Deliver Build Artifacts

Since AnthillPro does not know what artifacts you want captured, you will need to label them so they can be grouped and stored together. This is done by defining an artifact set on your build process. You can think of the artifact set as a grouping of build artifacts (files) that allows for fine-grained consumption of the artifacts. Once the artifacts are labeled and grouped into an artifact set, you will need to manually add a **Deliver Artifact** step to your build job.

To capture and deliver build artifacts:

1. Go to **Administration** and select the workflow you created in the Setting Up a Build section.
2. On the Workflow page, select the **Artifacts** tab, then click the **New Artifact Config** button.

3. Configure artifact set:

- **Artifact Set.** Select an artifact set from the drop-down menu (using one of the defaults is fine). AnthillPro ships with 3 default artifact sets that correspond to the most common tiers of a 3-tier web application:
 - **APP.** Used to group objects for deployment to the application tier of a 3-tier application.
 - **DB.** Used to group objects for deployment to the database tier of a 3-tier application.
 - **WEB.** Used to group objects for deployment to the web tier of a 3-tier application.

Generally, an artifact set is named for the type of objects which are grouped inside of them. The objects within an artifact set, in turn, are grouped by how the objects are going to be consumed. (For now, we are focusing on Artifact Sets used for deploying a project. In another section, we will cover using Artifacts Sets for dependency management.)

- **Base Directory.** Here, you need to give the directory where the artifacts (say a jar or dll file) are placed once the build is complete. This directory is relative to the build's working directory. So if your build places the artifacts in a "dist" directory, you would specify `dist/` here. Note that if you leave this field blank, AnthillPro will include the entire contents of the working directory in the artifact set.
- **Include Artifacts.** List the artifacts to be retrieved from within the base directory. You can specify the names of files that reside in the base directory: e.g., `myProjectArtifacts.zip`. Or, if the artifacts are located in a sub directory, you specify something like `bin/myProjectArtifacts.jar`. Each include pattern must be entered on a separate line.

You can also use the following wild cards to tell AnthillPro what to include in the artifact set:

- `**` Indicates include every directory within the base directory.
- `*` Used to include every file. So, if you use `*.zip`, the files matching this pattern will be included.
- `**/*` Tells AnthillPro to retrieve the entire file tree underneath the base directory.

If you leave this field blank, AnthillPro will include all the files in the base directory (if one was specified).

Advanced: To include symbolic links and empty directories, add the link/directory as part of your include pattern. See Configure Server Miscellaneous Settings.

- **Exclude Artifacts.** Give the patterns of the artifacts that should be skipped from the include. This field is set in the same way as the Include Artifacts field, only you are telling AnthillPro what NOT to include. If you leave this field blank, AnthillPro will exclude no files.
- Click **Save**.

4. To send the build artifacts to the management system (Codestation), you need to add the **Artifact Delivery** step to your build job. Go to the job configuration page of the job you created in the Setting Up a Build section.

5. Click the **Insert After** icon of the step prior to where this step should run. Remember, the steps run in order, so any step, such as the Artifact Deliver step, which deals with the build artifacts must come after the actual "build" step.

6. On the Steps page, expand the **Artifacts** folder, select **Artifact Deliver**, and click **Select**.

7. Configure step:

- **Name.** Give a name for this step. A simple "Artifact Deliver" is sufficient.

- **Artifact Set.** Select the artifact set you just finished configuring.
 - **Show Additional Options.** These are advanced settings. For a simple deployment, you can skip these settings.
 - Click **Save**.
8. If you configured more than one artifact set for the build, add an Artifact Deliver step for each set. For example, if you are collecting different artifacts into the APP, DB, and WEB artifact sets, then you will need three deliver steps -- one for each artifact set.
 9. The next time your build runs, the artifacts will be delivered to the AnthillPro artifact management system. (If you like, you can force a manual build.) This will make the artifacts available on the **Dashboard**.

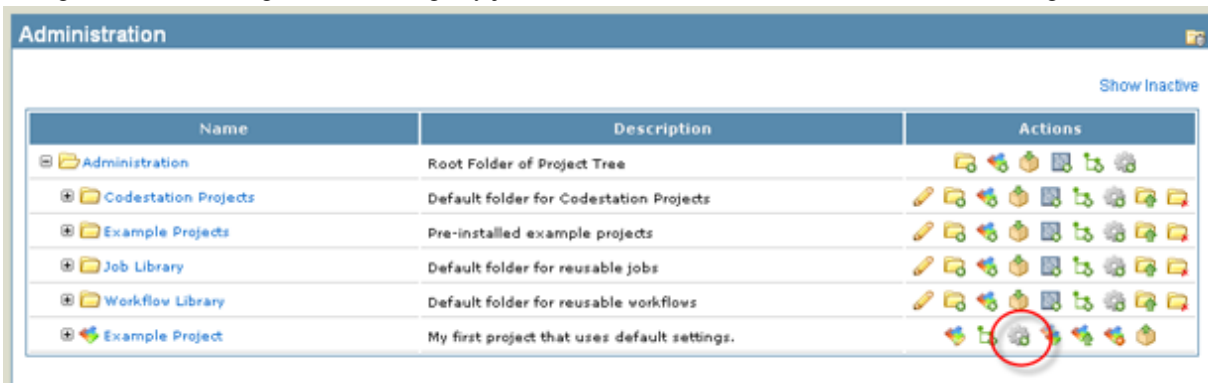
To view the artifacts, go to the Dashboard and select your build workflow. Click on the most recent **Build Life** (i.e., build number) and then select the **Artifacts** tab.

Once you have verified that the correct artifacts have been captured and delivered, see Create a Deployment Workflow. to send them to a different location.

Create a New Job

The job is the building block of the *transformation*, and is composed of a series of distinct actions the server must perform (called "steps" in AnthillPro) to successfully run a build. For any job you create, the steps (actions) are executed by the server one at a time, in a specific order. When you are first starting out with AnthillPro, it is best practice to use the Job Wizard when creating a build job. That way you are ensured that the server will perform the correct action at the correct time.

1. To create a job, click the **Add Job** icon of your project. Note that a job can only be created after the project's originating workflow has been created. If you get a message saying, "You must create an originating workflow and configure a source config before creating any jobs," see Create a New Workflow before continuing.



2. **Use the Job Wizard** to create your build job. Check **Yes** and click **Select**. The Job Wizard will take you through the steps necessary to create a build job.
3. **Set up the job.** On the Job's main page, you configure a number of controls, mostly dealing with how AnthillPro should interact with source. When possible, beginners should use the default values. For any job, configure the following:
 - **Name.** Give a name for this job. Typically, the name should reflect the type of work to be performed by the job. For example, "Build."

- **Description (optional).** Give a description of this job. The description allows you to convey more information about what this job does, etc.
- **Stamp Style.** Choose the DEV stamping style configured during workflow creation.
- For items such as **Should Tag, Should Cleanup, Change Log Start Status**, etc., use the default settings. These are advanced settings.
- Click **Next**.

4. **Add a builder.** On the **Builders** tab, click **Add a Builder**.

- For any job, you need to let AnthillPro know what builder you use when you build source code for this project. Choose your builder from the list below for instructions on configuring this item.

Ant	Maven	Ruby (Builder)
Groovy	MSBuild	Visual Studio (Builder)
Make	Nant	Shell/Perl Scripts

- Click **Next**.

5. Click **Done**. It is not necessary to add a publisher to complete a build. Publishers are used to share information about the completed build or send the build artifacts to Codestation, AnthillPro's artifact management system.

6. See Complete Build Configuration.

Creating a Custom Build Job

Using the Job Wizard to create a build job is recommended, as it is the best way to ensure that your builds will be configured correctly. However, it is also possible to create a custom build job. What this job will look like, and what steps it contains, will depend on your processes and tools used. Generally, every build job should contain the following steps at a minimum:

1. **Cleanup Step.** Cleans up the workspace prior to populating it. This ensures that any files you might pick up from the build actually belong to the build and not a previous build.
2. **Populate Workspace.** Places the checked-out code (defined in the workflow's source configuration) in the workspace.
3. **Get Changelog.** The retrieved changelog is usually based on the changes made since the previous build. This step enables AnthillPro to extract data from the SCM and then store it in the AnthillPro data warehouse. Since AnthillPro stores the changelogs, it can parse the data, allowing you to override the default behavior. For example, you can select a starting point for the changelog based on criteria such as the latest production build.
4. **Stamp.** Applies a stamp to the build, which is used to identify the build within AnthillPro. See Stamping for more.
5. **Get Dependency Artifacts.** If your project depends on any other project, you need to specify it here. Otherwise, this step will just be skipped if no dependencies are defined.
6. **Builder.** Usually points to the location of tool or script used for the build.

Ant	Maven	Ruby (Builder)
-----	-------	----------------

Groovy	MSBuild	Visual Studio (Builder)
Make	Nant	Shell/Perl Scripts

7. **Assign Status - Success.** Applies the status when the build completes successfully.

8. **Assign Status - Failure.** Applies the status when the build fails.

This build configuration uses generic job steps -- which should suffice for most builds. However, most of the SCM integrations also include tool-specific steps (commands, etc.) that you can add to your build job. While these steps are not required for a basic build, they may be helpful as you design more complicated build jobs.

When configuring a job, the steps are available under the SCM folder. The tool-specific job steps that may be added to the generic build job or used to replace one of the generic steps where appropriate. For example, the ClearCase integration includes the following tool-specific steps that can be used to either replace or append the generic build:

- **ClearCase Changelog.** Performs a ClearCase changelog. You can use this step in place of the standard step.
- **ClearCase Cleanup.** Perform a cleanup of the current ClearCase working directory. You can use this step in place of the standard step.
- **ClearCase Label.** ClearCase-specific label step. You can use this step in place of the standard step, if labels are being used.
- **Lock (ClearCase Base Dynamic Plugin only).** Locks ClearCase resources. This steps adds ClearCase specific functionality to the build job.
- **Unlock (ClearCase Base Dynamic Plugin only).** Unlocks ClearCase resources. This steps adds ClearCase specific functionality to the build job.
- **ClearCase Populate Workspace (Legacy integration only).** Populates the workspace With source from ClearCase. You can use this step in place of the standard step.

Complete Build Configuration

Once the job has been created, it needs to be added to the originating workflow you created earlier. This process ties the build process together, and allows AnthillPro to generate the build artifacts when the build is run.

1. Go to the workflow you created in the Create a New Workflow section.
2. Select the **Definition** tab. The Workflow Definition is used to define the execution of the Job used by this workflow. A Workflow Definition has a single starting point and a single end point.
3. Use the default **Embedded Definition**. Click **Select**.
4. Click the **Start** button and select **Insert Job After**. Configure:
 - **Job.** Select the **job** you created in the Create a New Job section. The job will appear in the drop-down menu under "Project Jobs."
 - **Pre-Condition.** Select **Always** as the pre-condition for this job. That way, the job will always run.
 - **Agent Filter.** Select at least one agent from the **Fixed Agent Selection** list. Highlight the agent name in the list

and click the plus sign (it will suffice to select one agent).

Once you are more familiar with AnthillPro, you can use agent filtering to determine which agent(s) a job can run on, including using scripted agent selection. For more on scripted agent selection scripts, see the scripting documentation in the Development Kit Bundle at [tools > anthill3-dev-kit.zip](#).

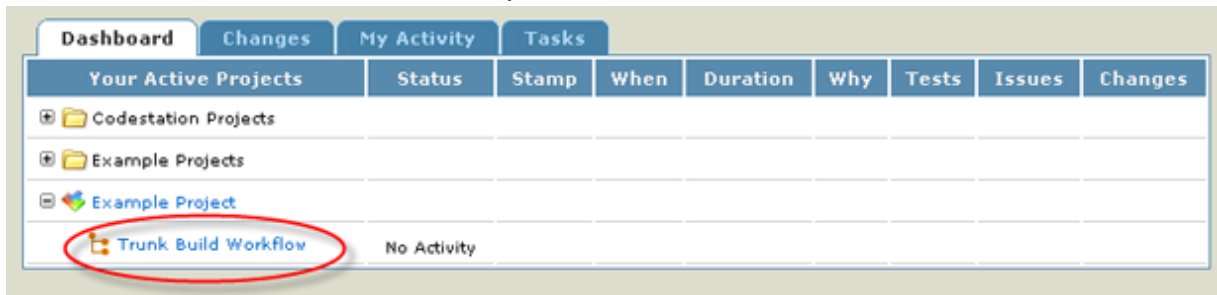
- **Working Directory.** Select **Source Config's Work Directory** from the menu. This will ensure that the build takes place in the correct location.
- **Lock for Workflow.** Use the default value of **No**. Locking is an advanced feature.
- Click **Insert Job**.

5. Now that you have configured the build process, the next step is to run the build. See [Run a Manual Build](#).

Run a Manual Build

Running a manual build is a simple push-button process. Once on the workflow's **Main** tab, you will be presented with build options. For now, we simply need to get the project building.

1. Go to the **Dashboard** and select the workflow you want to build.



2. On the workflow's Main page, Click the **Build** button.

- If you get a message saying "New Build Life Not Needed," make sure the **Force** option is selected and click the **Build** button again. Normally, AnthillPro only builds if the server finds source changes. Since this is the first build, you may have to force the build.
- **Delay Build (Advanced).** To delay a build, check the box and then give the date and time you wish this workflow to run. This is a one-time event, not to be confused with a scheduled build. Once you configure a delayed build, it appears on the **Current Activity** tab under the **Delayed Builds** (or **Planned Workflow Execution** in some versions of AnthillPro) menu. There, you can delete the request for a delayed build.

3. Once the build has been started, you can check the build status on the workflow's **My Activity** tab. Refresh the page to update status.

When done, you can consider some Next Steps.

Next Steps

- **Continuous Integration.** Now that you have the basic build going, try [Setting Up Continuous Integration](#).

- **Notifications.** You can configure AnthillPro to send out notifications about the state of your builds. See [Basic Notifications](#).
- **Setting up a deployment process.** Now that you have success with the basic CI build, you can turn your attention to the "output" phase of your build process: setting up an automated deployment process. See [Setting Up a Deployment](#).
- **Add a dependency.** Most project's will depend on another project. AnthillPro includes a dependency management system that allows you to use the build tool to define all your dependencies. See [Defining a Basic Dependency](#).

Chapter 11. Setting Up Continuous Integration

Once you have the basic build going and delivering the artifacts to Codestation (see [Setting Up a Build](#)), you are ready to create a Continuous Integration (CI) build process. To get the most out of CI, the build process can be supplemented with the following AnthillPro features:

- **Trigger.** When a trigger is activated for a build workflow, AnthillPro will automatically kick off a CI build when source changes are detected.
- **Notifications.** Send an e-mail to users when a workflow completes.
- **Test integration.** Integrate unit testing into your AnthillPro CI build workflow.

It's worth noting that you can add these "CI" features to your build process without having to practice CI. All the CI features either add automation or extend ease of use to every build.

Before you continue, make sure you have a consistent build going. If your project has a few successful builds, that will make it easier to troubleshoot problems as you add CI components to the build process (see [Setting Up a Build](#)). To start notifying team members about the state of the build, you will also need administrative access to your e-mail server.

Add a Workflow Trigger

Enabled on the workflow, a trigger is an automated mechanism for kicking off a process. In relation to CI, the trigger will allow for automated builds. In AnthillPro, you can use one of three trigger types to kick off a build:

- **Scheduled trigger.** The simplest type to configure, the scheduled trigger fires either on a regular interval (e.g., every few minutes, hours, etc.) or can utilize a Cron expression for irregular, but recurring, schedules. When used for CI, the scheduled trigger polls the SCM for source changes. If changes are found, a build is kicked off. See [Use Agent Filters and Quiet Periods](#) for more detailed information.
- **Repository trigger.** Many SCM types support commit triggers that allow AnthillPro to kick off a build at the time source changes are made. Repository triggers are far more efficient than scheduled triggers, as they don't poll the SCM, and are the most common trigger type used for CI builds. See [Use Agent Filters and Quiet Periods](#) for more detailed information.
- **Event trigger.** An advanced trigger type, the event trigger allows for a custom filter that taps into the AnthillPro event service and triggers actions when certain events occur. For example, an event trigger would listen for build completed events, check those events against the project's dependencies, and force a build of the project if a dependency builds successfully.

When setting up your first CI build in AnthillPro, using the "Every 15 minutes" schedule for your trigger is the simplest way to get the build going. The schedule was designed to be used in conjunction with the CI trigger.

To begin, activate a scheduled trigger:

1. Go to **Administration** and select the workflow of the project you created in the [Setting Up a Build](#) section.
2. On the Workflow page, select the **Triggers** tab and then click the **New Trigger** button.

3. Select **Scheduled Trigger** from the drop-down menu and click **Select**.
4. Configure trigger:
 - **Name.** Give a name that identifies this trigger. For example, "CI Trigger."
 - **Schedule.** From the drop-down menu, select the "Every 15 minutes" schedule that ships with AnthillPro. This is the default schedule to be used in conjunction with CI. Fifteen minutes is a good interval for CI, in that it allows for quick feedback and avoids unnecessary builds.
 - **Force.** By default, AnthillPro does not force a build if the trigger does not find any code changes when it polls the SCM. It is rare that you will want to force a build every time the trigger fires, so leave this field blank.
 - **Enabled.** By default, when you create a new trigger it is enabled, which is what you want for a CI build.
 - Click **Save** then **Done**.
5. The next time the trigger fires, and it detects a source change, the CI build will take place. It will be identified as a "Scheduled" build on the Dashboard.

Workflow	Status	Stamp	When	Duration	Why	Tests	Issues	Changes
Trunk Build Workflow	Complete	1.0.14	8/18/09 12:10 PM	27 s	Scheduled	0 / 0	0	0 / 0

Note that if you are not seeing scheduled builds, it is most likely because no changes have been made to the source code. To see a scheduled build, you need commit a source change and wait for the CI build to occur.

6. See Set Up Notifications.

Set Up Notifications (CI)

You can set up AnthillPro to notify users when a build completes, to provide developers with more feedback on the CI build. To do this, each workflow is assigned what is called a notification scheme in AnthillPro. The notification scheme defines:

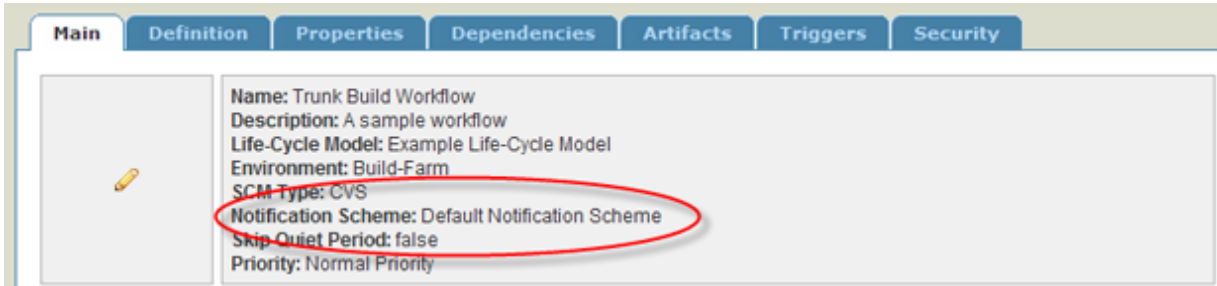
- **Who** is notified. For example, you can choose to notify just the developers who committed changes to the build, every user that works with the project, every AnthillPro user, etc.
- **When** a notification is sent. AnthillPro can be configured to send a message when a workflow status changes (i.e., the build breaks), when any workflow completes regardless if it fails or not, etc.
- **How** the notification is sent. Standard notifications are sent either by e-mail or instant message.

AnthillPro ships with a Default CI Notification Scheme (in some versions it is simply called Default Notification Scheme) that sends an e-mail to all repository committers upon a change in status to a CI workflow. Since you are just getting started, it will be easiest to use this notification scheme.

Since you will be setting up AnthillPro to send e-mails, you will need to do some basic configuration within AnthillPro and on your e-mail server. To begin sending notifications:

1. Follow the instructions given in the Configure Mail Server section and then go to the next item.
2. Go to **Administration** and select the workflow you created in the Create a New Workflow section.

3. On the workflow Main page, verify that you are using the **Default CI Notification Scheme**. If so, you are done. If not, go to the next item.



4. If the workflow is not using the Default CI Notification Scheme, click the **Edit** icon (the pencil on the left).
5. Select **Default CI Notification Scheme** from the Notification Scheme menu then click **Save**.
The next time a build is completed, committers to this project will receive an e-mail notification.
6. See Integrate Testing with Your CI Build.

Integrate Testing with Your CI Build

With AnthillPro automatically kicking off builds and sending out notifications, including a unit test suite that runs every time you run a build will help identify any decrease in code quality. If you use CppUnit, JUnit, or NUnit, AnthillPro has a built-in integration that will parse and then store your test results, making them available on the Dashboard. This will enable you to use AnthillPro to view results, track changes over time, and even have AnthillPro e-mail the results.

To integrate unit testing with your CI build, select the type of unit test you use from the list below and follow the instructions for adding them to your build process.

CppUnit	NUnit
JUnit	TestNG

When done, you can consider some Next Steps.

Next Steps

- **Switch to a repository trigger.** AnthillPro can be configured to use a repository trigger in conjunction with a SCM system. With a repository commit trigger in place, a build is kicked off every time changes are committed to the SCM. See Use Agent Filters and Quiet Periods and Workflow Triggers and Scheduled Builds.
- **Configure custom notifications.** Set up custom notifications with AnthillPro based on a number of different criteria, including e-mailing test results. See Managing Notifications.
- **Setting up a deployment workflow.** Now that you have success with the basic CI build, you can turn your attention to the "output" phase of your build process: setting up an automated deployment process. See Setting Up a Deployment.

Chapter 12. Basic Notifications

Once configured, AnthillPro can routinely run builds with no interaction with the development or build team. However, if it fails to communicate the results of builds, deployments, and promotions back to the team, it is only useful when people log-in to the system to manually check the status. That is not very good. A better model is to have AnthillPro send e-mails, instant messages, and other notifications to select team members.

The fundamental unit for managing notifications sent by AnthillPro is the Notification Scheme. A notification scheme sets rules determining what groups of users are sent which kind of notification about specified events. Each workflow is configured with a notification scheme within AnthillPro. The same scheme may be shared by many workflows, even workflows in different projects.

AnthillPro sends a number of notifications to users in a variety of formats. Most commonly, AnthillPro is configured to send either an e-mail or IM message regarding the state of a CI build. The recipient list of these notifications, as with other notifications AnthillPro sends out, are usually tied to the LDAP integration, etc., which is configured as part of AnthillPro's security system; as well as your mail server and/or IM provider. This allows you an easy way to send notifications to AnthillPro users (it is also possible to send notifications to non-users as well). You can even compose custom notifications that send the information you want in the format you want. Currently, you have the following standard options:

- **Email notifications.** The fundamental unit for managing notifications sent by AnthillPro is the Notification Scheme. A notification scheme sets rules determining what groups of users are sent which kind of notification about specified events. Each workflow is configured with a notification scheme within AnthillPro. The same scheme may be shared by many workflows, even workflows in different projects. See [Managing Notifications](#).
- **Instant Messaging.** AnthillPro can be used to send instant messages using Google Talk, Jabber, or MSN IM. For example, AnthillPro may be configured to send an instant message to a group of committing developers when a build fails, etc. See [Managing Notifications](#).

The Google Talk and Jabber integrations are configured by following the XMPP IM link under the Notification menu on the System page. The integration can be disabled any time by clicking the **Disable** button. Using the **Test User ID** and **Service Status** fields allow you to troubleshoot the connection.

In addition to sending notifications via e-mail and IM, administrators can send out announcements to all AnthillPro users. Once an announcement is sent, it will appear at the top of the UI for all AnthillPro users. Common uses for announcements are to let everyone know when the server is going down for maintenance, report network problems, etc. See [Managing Notifications](#).

Set Up Basic Notifications

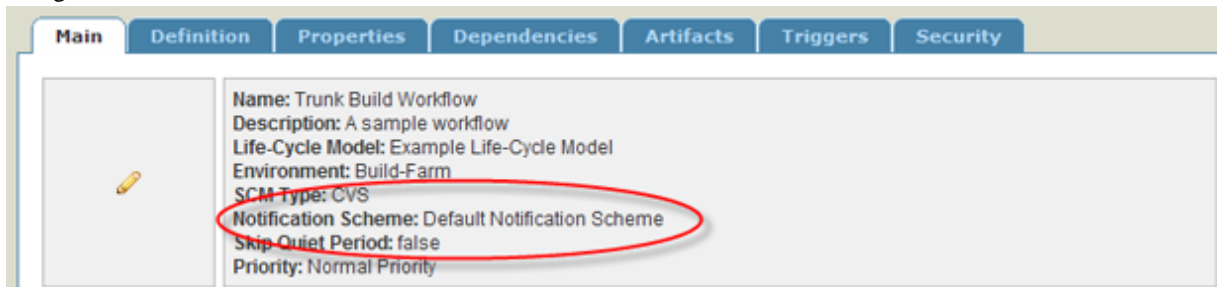
Each workflow is assigned what is called a notification scheme in AnthillPro. The notification scheme defines:

- **Who** is notified. For example, you can choose to notify just the developers who committed changes to the build, every user that works with the project, every AnthillPro user, etc.
- **When** a notification is sent. AnthillPro can be configured to send a message when a workflow status changes (i.e., the build breaks), when any workflow completes regardless if it fails or not, etc.
- **How** the notification is sent. Standard notifications are sent either by e-mail or instant message.

If you followed the instructions for setting up a CI build, you have already configured the basic notifications. This section is intended for those who are not configuring CI builds.

AnthillPro ships with a Default Notification Scheme that sends an e-mail to all repository committers upon a change in status to a workflow. We'll use this as a basic notification. To begin sending notifications:

1. Follow the instructions given in the Configure Mail Server section and then go to the next item.
2. Go to **Administration** and select the workflow you want to add notifications to.
3. On the workflow Main page, verify that you are using the **Default Notification Scheme**. If so, you are done. If not, go to the next item.



4. If the workflow is not using the Default Notification Scheme, click the **Edit** icon (the pencil on the left).
5. Select **Default Notification Scheme** from the Notification Scheme menu then click **Save**.

The next time a build is completed, committers to this project will receive an e-mail notification.

Chapter 13. Setting Up a Deployment

If you recall, the build process is composed of three stages: input, transformation, and output. Once you have a consistent build going, you should have a pretty good idea on how AnthillPro tackles the first two phases of the build process to generate output and deliver the artifacts to Codestation (the artifact management system). At this point, you can set up a deployment process in AnthillPro that will enable you to move the artifacts (or the output of the build process) to a different location. For example, the deployment process is typically used to move the artifacts so they can be: tested in a different environment; delivered to specialized hardware; subjected to exhaustive testing; etc. Once the deployment process is set up, AnthillPro will keep track of what was deployed; when the deployment took place; where the artifacts ended up; who ran the deployment; as well as the build that generated the artifacts.

Setting up a deployment process is similar to setting up the build process: you will need to configure a workflow and a job. However, since a deployment is a secondary process (called a non-originating workflow in AnthillPro) that runs after a build has completed, you will need to configure what is called a non-originating workflow. In other words, a non-originating workflow is configured for any process you want to subject an existing build to. So a deployment is one type of secondary process responsible for moving the artifacts around.

To achieve a successful deployment, you will need to:

1. Already have at least one successful build in AnthillPro. It's a good idea to get a consistent build process in place before dealing with deploying artifacts. If you have not successfully built a project in AnthillPro, see [Setting Up a Build](#).
2. **Create a deployment workflow.** Creation is similar to that of a build workflow, but is simpler to configure. A deployment workflow (implemented as a non-originating workflow) inherits many of its properties from the project's build workflow.
3. **Create a deployment job.** Similar to creating a build job, the deployment job includes steps that move the artifacts to a different location. However, you will not use the Job Wizard to create the deployment job.
4. **Run a deployment.** Once configuration is done, you are ready to run the deployment.

Create a Deployment Workflow

Remember that in AnthillPro the originating workflow is used for setting up a build process, and that the non-originating workflow is set up to run a secondary process on a completed build. So when setting up a deployment process, you will need to configure a non-originating workflow.

To create a deployment workflow:

1. Go to **Administration** and select the **Add Workflow** icon of the project you created in the [Setting Up a Build](#) section.
2. Next, check **Non-originating** workflow and click **Select** on the New Workflow page. In AnthillPro, it is the non-originating workflow that is used to run a deployment.
3. **Workflow configuration:**
 - **Name.** Give a name for this workflow. Typically, the name should reflect the type of work to be performed when this workflow runs. For example, "Deploy."
 - **Description (optional).** Give a description of this workflow. The description allows you to convey more information about the workflow.

- **Environment.** In AnthillPro, the environment is the location where the deployment takes place. Select the environment you would like this deploy workflow to run in. For example, if your build took place in the Build-Farm and you want to deploy it to the QA environment, select the QA environment from the drop-down. If you recall, AnthillPro ships with 3 default environments:
 - **Build-Farm.** Generally, this is the environment that all builds take place within, and is most closely associated with Development. This is the environment you want your builds to run in.
 - **QA.** This environment is provided for performing quality assurance testing. Most often, when you are sending a build to testing, you want that workflow to run in this environment. This is the environment you want to use when running a secondary process (i.e., non-originating workflow) that deploys the artifacts to your testing environment.
 - **Production.** This environment is typically used to deploy your builds once they are ready for production.
- **Notification Scheme.** Select the **Default Notification** scheme. By default, AnthillPro will send an e-mail notification to AnthillPro users. For now, don't worry if you aren't getting any e-mails. Notifications are covered in more detail in the Basic Notifications section. If you have not already done so, you may want to configure the notifications for this and other workflows.
- **Priority.** Use the default Normal Priority. Once you have a lot of projects going in AnthillPro, you may need to prioritize which workflows run first.
- Click **Save**.

4. See Create a Deployment Job.

Create a Deployment Job

The job is composed of a series of distinct actions the server must perform (called "steps" in AnthillPro) to successfully run a deployment. For any job you create, the steps (actions) are executed by the server one at a time, in a specific order. When configuring the deployment process, you will need to manually create a job.

Every deployment job must:

- **Set and clean the working directory.** To ensure that a clean deployment is performed every time, this step will allow AnthillPro to run a clean-up at the beginning of the job.
- **Resolve the artifacts.** This step retrieves artifacts generated by the originating workflow and deposit them in a specified directory.
- **Deploy the content.** This step uses a builder to run a deploy script. For example, if you use Ant, you would select it from the Builders menu and then configure the script.

To configure the deployment job:

1. Go to **Administration** and select the **Add Job** icon of the project you created in the Setting Up a Build section.
2. Check **No** and click **Select**. Do not use the Job Wizard.
3. Set up job:
 - **Name.** Give a name for this job. Typically, the name should reflect the type of work to be performed by the

job. For example, "Deploy."

- **Description (optional).** Give a description of this job. The description allows you to convey more information about what this job does, etc.
- Click **Set**.

4. On the **Job Configuration** page, click the **Create Step** button.

5. Next, add a Set Working Directory step. Expand the **Miscellaneous** folder, select **Set Working Directory**, and then click **Select**.

- **Working Directory Script.** Select the Default Project Working Directory script from the drop-down. For most projects, the default script will be sufficient.
- **Clean Working Directory.** Check this box. This will erase files and subdirectories in the Working Directory when this step runs, thus ensuring that a clean deployment will take place.

If you did not use a default working directory script above, you will need to verify that the working directory is not set to something like `C:` or `C:\` because **the entire contents of the C drive will be permanently removed** when the Clean Working Directory step is run.

- **Show Additional Options.** These are advanced settings. For a simple deployment, you can skip these settings.
- Click **Save**.

6. **Add the Resolve My Artifacts step.** This step will retrieve the artifacts generated by the build you configured earlier. Click the **Insert After** icon (under the Actions Menu) of the Set Working Directory step, expand the **Artifacts** folder, select **Resolve My Artifacts**, and click **Select**.

- **Name.** Giving a simple name of "Resolve Artifacts" is sufficient.
- **Artifact Set.** Select the artifact set you configured in the Capture and Deliver Build Artifacts section during basic build job creation.
- **Directory.** Specify what directory you want artifacts delivered to. This is relative to the working directory. For example, if this workflow's working directory is `C:\myProject\deployWorkflow\`, and you specify "artifacts" here, the artifacts will be delivered to `C:\myProject\deployWorkflow\artifacts`.
- **Transfer Only Changed Files.** Check here if you want AnthillPro to transfer only changed files based on checksums. For configuring your first deployment, this optional feature is unnecessary. See also Server Security.

If you are using the **Transfer Only Changed Files** option for your Resolve Artifact step, you must activate the **Digest Algorithm** server setting. Once set, AnthillPro will use either a SHA or MD cryptographic hash function to protect the build and deployment artifacts. AnthillPro will then use the generated file checksums for the resolve, and skip the Codestation caching on the agent. Use of this feature can result in faster deployments that use less bandwidth -- especially if the deployment target is a remote server and only a small proportion of the files will change between deployments.

- **Include Patterns.** List the artifacts to be resolved. If you want to include all the artifacts, simply leave this field blank, (or use the wild card `**/*`). You can refer to the Capture and Deliver Build Artifacts section for additional information on using include patterns.
- **Exclude Patterns.** Give the patterns of the artifacts that should be skipped from the include. This field is set in the same way as the Include Artifacts field, only you are telling AnthillPro what NOT to include. If you leave this field blank, AnthillPro will exclude no files. You can refer to the Capture and Deliver Build Artifacts sec-

tion for additional information on using exclude patterns.

- **Show Additional Options.** These are advanced settings. For a simple deployment, you can skip these settings.
- Click **Save**.

7. If you configured more than one artifact set for the build, add the Resolve and step for each set. For example, if your build published different artifacts into the APP, DB, and WEB artifact sets, then you will need three resolve steps -- one for each artifact set.

8. **Add a Deploy Artifacts step.** This step will deliver the resolved artifacts to the target destination. To do this, you will have to configure a **Builder** step that will run your deploy script. Most users simply use the same tool for builds and deployments. To configure this step:

- Click the **Insert After** icon (under the Actions Menu) of the Resolve My Artifacts step, and then expand the **Builders** folder.
- You will need to select one of the following and then click **Select**:

Ant	Maven	Ruby (Builder)
Groovy	MSBuild	Visual Studio (Builder)
Make	Nant	Shell/Perl Scripts

- Follow the instructions given for the builder you are using to run your deploy script.

Note that when configuring the step, give it a name such as "Deploy Artifacts" so that it can be differentiated from other builder steps. Also, if you see a **Script Content** tab, that means you can write your script directly in the AnthillPro UI.

9. When you have added the Deploy Artifacts (Builder) step, see Complete Deployment Configuration.

Complete Deployment Configuration

Once the deployment job has been created, it needs to be added to the deployment workflow you created earlier. This ties the deployment process together, and allows AnthillPro to send the artifacts to the desired location.

1. Go to the workflow you created in the Create a Deployment Workflow section.
2. Select the **Definition** tab. The Workflow Definition is used to define the execution of the Job used by this workflow. A Workflow Definition has a single starting point and a single end point.
3. Use the default **Embedded Definition**. Click **Select**.
4. Click the **Start** button and select **Insert Job After**. Configure:
 - **Job.** Select the **job** you created in the Create a Deployment Job section. The job will appear in the drop-down menu under "Project Jobs."
 - **Pre-Condition.** Select **Always** as the pre-condition for this job. That way, the job will always run.
 - **Agent Filter.** Select at least one agent from the **Fixed Agent Selection** list. Highlight the agent name in the list and click the plus sign (it will suffice to select one agent).

Once you are more familiar with AnthillPro, you can use agent filtering to determine which agent(s) a job can run on, including using scripted agent selection. For more on scripted agent selection scripts, see the scripting documentation in the Development Kit Bundle at *tools > anthill3-dev-kit.zip*.

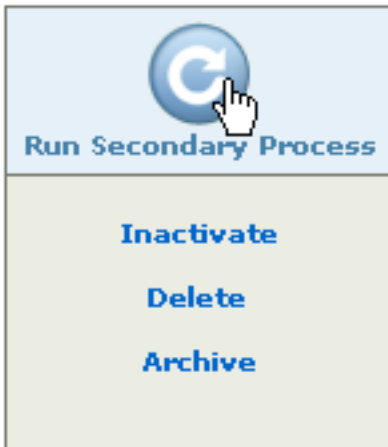
- **Working Directory.** Select **Source Config's Work Directory** from the menu. This will ensure that the build takes place in the correct location.
 - **Lock for Workflow.** Use the default value of **No**. Locking is an advanced feature.
 - Click **Insert Job**.
5. Now that you have configured the deployment process, the next step is to run the deployment. See Run a Deployment.

Run a Deployment

Running a manual deployment is a simple push-button process performed on the Dashboard. Once you select a build number (called Build Life in AnthillPro), the build's Main page gives you an option to run a secondary process (i.e., run the deployment workflow you just set up).

To run your deployment:

1. Go to the **Dashboard** and select the build workflow of your project.
2. On the workflow's **Main** page, Select the Build Life number of one of your builds.
3. On the Build Life page, click the **Run Secondary Process** button.



4. Next, run the secondary process (i.e., deployment workflow):
 - **Workflow.** Unless you have configured multiple non-originating workflows for this project, AnthillPro will automatically populate this field.

If you are given a drop-down, select the workflow you created in the Create a Deployment Workflow section and then click **Next**.

- **Environment.** AnthillPro will automatically populate this field for you if you only check one environment while configuring the deployment workflow.

If you are given an option, select the environment you want this workflow to run in. For example, if you want it to run in a QA environment, select it here.

- **Delay.** You can leave this blank. That way, the deployment will run immediately.

Advanced: To delay a deployment, check the box and then give the date and time you wish this workflow to run. This is a one-time event, not to be confused with a scheduled deployment. Once you configure a delayed deployment, it appears on the **Current Activity** tab under the **Delayed Builds** (or **Planned Workflow Execution** in some versions of AnthillPro) menu. There, you can delete the request for a delayed deployment.

- Click **Run**.

5. Once the deployment has been started, you can check the status on the **Build Life** page (the page you were returned to). Refresh the page to update status. When the deployment is completed, it will be listed on the Build Life page, along with the build (originating workflow).

It's worth noting here that the same deployment process can be run on any build (Build Life). For example, if you have 3 successful builds, you can run the same deployment process on each Build Life.

Also, there is no limit to how many times a secondary process can be run on a build (Build Life). So, you can run the same deployment as many times as you want for any given build.

AnthillPro will always keep track of which deployment belongs to which Build Life (build), and how many times the deployment was run on each particular build.

Set Up Automated Deployment

Once you have your basic deployment workflow running, you may want to automate the deployment instead of having to click the deploy button every time. There are a number of ways to do this, but following are the two most common scenarios.

- **Add a "deploy" job to your build.** If you want to practice continuous deployment, appending your build job with a "deploy" job -- which contains a single step: **Run Another Workflow** -- will deploy the content every time a build is successful. It's worth noting that this option can become resource intensive if used on every project. To auto deploy upon build completion:

1. Ensure that your deployment workflow successfully runs. If it you have not created one yet, go to the beginning of this section and create one. You will also need to ensure that your build workflow is delivering the artifacts to AnthillPro's artifact management system.
2. Go to **Administration** and then select the Project that you want to deploy.
3. Click on the **Add Job** icon. Do not use the Job Wizard.
4. **Name** the job ... something like "Send to [Environment]" and give a description. It's a good idea to give this job a different name than the job used by the actual deploy workflow.
5. Insert a single step in the job: **Run Another Workflow**. This step is contained in the Miscellaneous folder. Configure the step:

- **Name** the step.

- **Description.** Give an optional description.
 - **Workflow.** Select the deploy workflow you want to run.
 - **Environment.** Select the environment you want this workflow to run in. This will typically be the environment that your deploy workflow runs in.
 - **Wait for Workflow.** Check the box if you want AnthillPro to wait until the workflow completes before running the deploy workflow. For this scenario, you should be able to skip this option. (If any of them fails, then this step will be failed) Caution: While waiting, this workflow will still hold any Lockable Resource, and this job will count as running on an agent.
 - **Pass Properties.** You have the option of passing properties from the build workflow to the deploy workflow. Under this scenario, you can select **Do Not Pass**. However, if you need to pass all the properties, select **Pass All**, or if you need to pass only matching properties -- properties that are set on both workflows -- use that option.
 - **Show Additional Options (optional; advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
6. Click **Save**.
 7. Next, still on the **Administration** page, go to the **originating (build) workflow** and click the **Definition** tab.
 8. Click on the **Stop** icon (at the bottom) and select **Insert Job Before**. If you have more than one job in the definition, be sure you add the deploy job to the end. Then set the job configuration:
 - **Pre-Condition.** Here, if you select the default option "All Ancestor Jobs Success or Not Needed" then this job will run (i.e., kick off the deployment) when the build job is successful. This option will only kick off a deployment upon build completion. The "not needed" portion comes into play if you have multiple jobs in the build definition but not all of them run all the time.
 - **Agent Filter.** Here, if you select the same options as you did for your build job everything should work fine. Since the job is simply calling another workflow, nothing much is going on here that requires a specific agent.
 - **Working Directory.** Selecting **Parent Job's Working Directory** is the simplest option. As with the Agent Filter, nothing much is going on here to be concerned about.
 - **Lock for Workflow.** Typically, selecting **No** will suffice. Unless you start having locking issues or have multiple jobs attempting to run the same working directory at the same time, it is unnecessary to lock for the workflow.

9. Save the your work.

The next time your project builds, it will automatically kick of the deploy workflow.

- **Add a scheduled trigger to your deploy workflow.** This method is most commonly used when you want to run a deployment on a regular schedule (e.g., deploy the most recent build to a testing environment during down time).

To do this, go to the deploy workflow, select the trigger tab and then choose Scheduled Trigger. A basic interval schedule can be used if you are deploying a nightly build, or if you have a more complex scenario, you can use a cron schdule. See Workflow Triggers and Scheduled Builds for more.

Next Steps

- **Integrating with other tools.** Now that you have a successful build-and-deploy process going, you can add further automation to your processes by using AnthillPro's robust integrations. See Integrating Other Tools.
- **Setting up dependencies between projects.** Use AnthillPro's unique dependency management system that lets you set up relationships and provide visibility into your dependency structures. See Dependency Management.

Chapter 14. Defining a Basic Dependency

Dependencies are configured on originating (build) workflows so that the artifacts of the child project can be retrieved at run time. Codestation, AnthillPro's dependency-management system, enables you to set how dependency builds occur. For example, a dependency can be configured so that the parent project checks its dependencies and builds them at run time. This is called a pull build. Or, a build for the child project can kick off a build of the parent. This is called a push build.

It is recommended to review the [Configure Dependencies](#) section for more -- especially when configuring the conflict strategy.

To successfully configure a dependency relationship requires configuration on both the parent and child projects:

1. **Artifact Set configuration on the child project.** AnthillPro stores the build artifacts of the child project in Codestation as an Artifact Sets.

Before you can complete a dependency relationship, AnthillPro will need to capture the child project's build artifacts and then deliver those artifacts to the embedded dependency-management system (called Codestation). Once the artifacts are in Codestation, AnthillPro will be able to keep track of when and where they are used as dependencies.

Since AnthillPro does not know what artifacts you want captured, you will need to label them so they can be grouped and stored together. This is done by defining an artifact set on your build process. You can think of the artifact set as a grouping of build artifacts (files) that allows for fine-grained consumption of the artifacts. Once the artifacts are labeled and grouped into an artifact set, you will need to manually add a Deliver Artifact step to your build job.

See [Capture and Deliver Build Artifacts \(Child Project\)](#) for more.

2. **Dependency configuration on the parent project.** Once the child project's build artifacts are captured and delivered to Codestation, the dependency is configured on the parent project's originating workflow. You will also need to add a Resolve Dependency Artifacts step to the parent project's build job.

Once the child project's artifacts are in Codestation, you need to configure the dependency relationship on the parent project's originating (build) workflow, and add a resolve dependency step to the build job. When configuring dependencies, you will also need to tell AnthillPro how you want your dependencies handled.

See [Configure Dependency \(Parent\)](#) for more.

Chapter 15. Integrating Other Tools

AnthillPro ships with a number of built-in integrations with many of the tools used throughout the project's lifecycle. The integrations, which are implemented at the job and step levels, enable AnthillPro to distribute process execution and invoke third-party tools. In addition to driving other tools, the integrations allow AnthillPro to act as an information hub: enabling AnthillPro to exchange information with other tools, and then perform actions based on the information.

If AnthillPro does not currently integrate with a tool you use, you can either contact us at <sales@urbancode.com> or you try writing your own integration using the Plugins feature.

Tool Integrations

AnthillPro currently supports integrations with the following tool types:

- **Source Code Management Tools.** Enables AnthillPro to check out code, access the changelog, and label the repository (where supported). See SCM Tools.
- **Build Tools.** AnthillPro integrates with many build tools to support building and deploying. See Build Tools.
- **Testing Tools.** Allows you to run a suite of automated tests either on your build workflow or as a secondary process -- in addition to gathering data from tests that are invoked with your build script. See Testing Tools.
- **Coverage Tools.** Collects information on the coverage tests that you invoke with your build script. Enables you to perform trending over time, see which tests are failing, track coverage, and even fail a build based on the captured results. See Coverage Tools.
- **Source-code Analysis Tools.** You can have AnthillPro run your source-code analysis tool and then collect the results. The integrations also allow AnthillPro to collect the test output and store it in the AnthillPro data warehouse. See Source-code Analysis Tools.
- **Issue Tracking Tools.** Enables you to associate an issue with a particular build, link to that issue in your issue tracker, and view the status of the issue. The integrations also allow AnthillPro to perform tasks within the issue tracking tool. See Issue Tracking Tools.
- **Virtualization.** Allows AnthillPro to clone, deploy, and undeploy configurations. Implemented as job steps, the integration can be used as part of your build workflows or as a secondary process. See VMware Lab Manager.

Plugins

With Plugins, you can write your own integration with third-party tools (such as testing, SCM, source-code analytic, etc., tools) and then add it to your AnthillPro processes. If you are at least minimally familiar with programming, you can easily write your own custom integrations. See Using AnthillPro Plugins.

Other Integrations

AnthillPro can be configured to use your security and notification systems, as well as with an IDE:

- **Security.** AnthillPro enables you to use your existing server for authorization and authentication, and to map existing users to roles in AnthillPro. See Setting Up Security.

- **Notifications.** AnthillPro notifications use your current tools to send a variety of notifications. See Notifications.
- **IDE Plugins.** The Plugins enable you to view the current activity and state of your projects, kick off a new build, map your projects to projects in AnthillPro, and to retrieve the project's dependency artifacts -- all from the IDE. See the Developer Kit on the AnthillPro Tools page (go to Tools > **Developer Kit Bundle**).

Part III. Upgrade

It is a best practice to back up your AnthillPro server before upgrading. The minimum suggested backup is to backup the database. If you are using the embedded Apache Derby database, then a backup can be done through the AnthillPro Web UI on the System tab by following the Backup Settings link under the Server menu. Click the Backup Now button. The backup will take a few moments.

- To backup other databases, please follow the procedures outlined by the respective database vendor. See Configure Backups of AnthillPro before continuing.

Once you have backed up the database, upgrade the server before upgrading the agents, distributed servers, or agent relay.

Chapter 16. Server Upgrade

Before running the upgrade script, you should shut down the server. To perform an upgrade, follow the installation instructions. See Installation.

*When using SSL, you need to turn it on for both the **server and the agent** -- otherwise the agents will not be able to connect to the server. If you are using the agent relay, ensure that **SSL is turned on for all three components: server, agent, and agent relay**. This rule also applies if using mutual authentication.*

- If you are planning to migrate databases as part of the upgrade, see Migrate Server Database before continuing.

When asked for the directory to install the server, enter the existing server directory.

To upgrade the AnthillPro Server:

1. Follow **steps 1 to 5 of the installation** process.
2. When prompted, give the current AnthillPro **server** directory. For example: **C:\Program Files\anthill3\server** (Windows) or **/opt/anthill3/server** (Linux).
3. The installer will ask you if you want to upgrade the existing server. Type **Yes** to upgrade or **No** to exit the installer. Once the upgrade is under way, the installer will use most of your existing values.
4. Provide the **Home directory of the JRE/JDK** used to run the server, if different from the original.
5. If the AnthillPro server is installed as a **Windows service**: Input the **unique name** in the dialog box and click **OK** to stop and remove service (Windows service will be recreated, if desired, at the end of the upgrade process).
 - If more than one instance of the AnthillPro server is installed as a Windows service, make sure the correct service name is entered.

See also Install as Windows Service.

6. Once the installer is complete, you can start the server.

Chapter 17. Migrate Server Database

We have seen some scalability issues with the embedded Apache Derby database. If you experience slowness in the AnthillPro Web UI or if the server process is taking up too much memory, it is likely time to migrate to a different database.

The AnthillPro command line installers include a database migration utility that exports your existing database data and imports it into another supported database.

If you want to move or create a test instance of your AnthillPro server, *but keep the same database type*, see Clone AnthillPro Instance.

The following databases are supported using the migration utility:

- Oracle. Use the JDBC driver for Oracle 10.2 or higher, regardless of what version of the database is used.
- MySQL with InnoDB storage (works with 4.1.22 and later). If using MySQL 5, use the 5.0.8 driver version. The 5.1 version has some bugs that will cause the AnthillPro server to throw an error. If you are using the 5.1 version, switch the driver jar file in the server's lib/ext directory and then restart. It is also recommended that the InnoDB [<http://www.innodb.com/>] storage engine be used.
- Microsoft SQL Server. The MS SqlServer 2.0 driver (sqljdbc4.jar) will not work with AnthillPro. To install the server using Microsoft SQL Server, you will need to use one of the following drivers, depending on which version of Java the AnthillPro server uses:

Java 5. Use either the 1.2 driver or the 2.0 sqljdbc.jar driver.

Java 6. Use the 1.2 driver.

- DB2 (requires 9.7 or later).
- PostgreSQL. Due to a known defect with PostgreSQL 8.4.0, AnthillPro can't be installed with that version. Use **version 8.3.7**.

See the Installation for migrating server database.

Database Requirements

These requirements apply if you want to use a database other than Apache Derby:

1. You must use one of the command line installers for installation.
2. You need to download the appropriate JDBC driver file for your database. These are typically downloaded from the database vendor.
3. You need to create an empty database for AnthillPro to use with a dedicated user.
 - The database migration should only be run after you have completed an upgrade to the version you are using to migrate. So if you download a new version and want to migrate, you must first do an upgrade.

Windows Database Migration

To migrate the AnthillPro database:

1. Shut down the server.
2. Download the **anthill3-<version>.zip** file.
3. **Expand** the zip file using a tool like WinZip. Expanding will create an **anthill3-install** directory.
4. Open the **anthill3-install** directory created in the previous step in **Windows**.
5. Copy the **JDBC driver file(s)** of the database you are migrating to into the **anthill3-install\lib\ext** directory.
6. Run the migration script **migrate-db.bat**.
7. If the AnthillPro server is installed as a **Windows service**: Input the **unique name** in the dialog box and click **OK** to stop and remove service (Windows service may be recreated, if desired, in step 10).
 - If more than one AnthillPro server is installed as a Windows service, make sure the correct service name is entered.
8. Provide the **directory** where the AnthillPro server is installed.
9. Input **Y** to **upgrade** the existing server.
- 10 Provide the following:
 - **Database type** AnthillPro should migrate to [derby, oracle, mysql].
 - **Database driver**. See your database vendor documentation for the JDBC driver class to use.
 - **Database connection string**. See your database vendor documentation on the format of this value.
 - **Database user name**.
 - **Database password**.
- 11 Follow the remainder of **step 7 through step 10 of the Windows installation** process. See Windows Installation.

Linux/Unix Database Migration

To migrate the AnthillPro database, follow the steps below:

1. Shut down the server.
2. Download the **anthill3-<version>.tar.gz** file.
3. Open a **UNIX shell** to the directory containing the above downloaded file.
4. **Extract** the downloaded **tar.gz** file. Type `tar -zxf anthill3-<version>.tar.gz`.

On some installations of Solaris and HP-UX the default tar command will not properly handle our tar files. You may need to use `\ install GNU tar`. When installing AnthillPro on Solaris, it is recommended to use korn shell (ksh).

5. **cd anthill3-install.**
6. Copy the **JDBC driver file(s)** of the database you are migrating to into the **anthill3-install/lib/ext** directory.
7. **Run** the migration script **./migrate-db.sh.**

During the migration you will be prompted for the following information:

1. **Directory** where the AnthillPro **server** is installed.
2. **Database type** AnthillPro should migrate to (`derby`, `oracle`, `mysql`, `etc.`).
3. **Database driver.** See your database vendor documentation for the JDBC driver class to use.
4. **Database connection string.** See your database vendor documentation on the format of this value.
5. **Database user name.**
6. **Database password.**

Chapter 18. Agent Upgrade

You should upgrade the server before upgrading any agents. AnthillPro supports an auto-upgrade feature to upgrade running agents.

The upgrade can be performed through the AnthillPro Web UI at **Agents > Agent**. Select the agents you want to upgrade and then click the Upgrade Agents button to begin the auto-upgrade process. The upgrade can take some time, so refresh the page to view progress.

- Because of extensive agent-server communication changes made for version 3.6, if you are upgrading to AnthillPro 3.6 the following applies: **Communication port**. A new port needs to be opened to the server (default value of this port is 7915) for agent communication. **Windows service (upgrading from 3.5 to 3.6.1)**. AnthillPro will attempt to upgrade agent Windows services and restart them after upgrade. However, if the agent service is running as a user that does not have permission to modify the service settings, the Windows service will need to be removed and reinstalled.

If you need to manually upgrade an agent, stop the agent and follow the same instructions as an installation. When asked for the directory to install the agent, enter the existing agent directory.

1. Shut down the server and agent.
2. Follow **steps 1 to 4 of the installation** process.
3. Give the **Directory** where the AnthillPro **agent** should be upgraded.
4. Input **Y** to **upgrade** an existing AnthillPro agent.
5. Give the home directory of the **JRE/JDK** used to run the agent.
6. **Relay connection**. Enter the default **N** if *not* using UrbanCode's Distributed Servers product, available under separate license. If using Distributed Servers, see Installing Distributed Servers before continuing.

If you are interested in using the Agent Relay for agent-server communication, contact [<info@urbancode.com>](mailto:info@urbancode.com).

7. Determine if internal communication should use **SSL**. Default is **N**.

*When using SSL, you need to turn it on for both the **server and the agent** -- otherwise the agents will not be able to connect to the server. If you are using the agent relay, ensure that **SSL is turned on for all three components: server, agent, and agent relay**. This rule also applies if using mutual authentication.*

- If using SSL, determine if communication should be **mutually authenticated**. Default is **N**.
8. If the AnthillPro agent is installed as a **Windows service**: Input the **unique name** in the dialog box and click **OK** to stop and remove service (Windows service will be recreated, if desired, at the end of the upgrade process).
 - If more than one AnthillPro agent is installed as a Windows service, make sure the correct service name is entered.

See also Install Agent as Windows Service.

Chapter 19. Distributed Servers Upgrade

Beginning with version 3.6.3, AnthillPro provides an upgrade path for Distributed Servers. If you are using an older version (i.e., 3.6.2 or older), you will need to install version 3.6.3 in order to perform future upgrades.

1. Before performing the upgrade, stop Distributed Servers and backup the Distributed Servers directory.
2. Run the install script in the Distributed Servers installation package.
3. When asked for the directory, give the directory the old version is installed in.
4. The installer will ask you if you want to upgrade. Answer **Y**.
5. Follow the instructions for an installation.

See also [Agent Relay Upgrade](#) and [Installing Distributed Servers](#).

Chapter 20. Agent Relay Upgrade

Beginning with version 3.6.3, AnthillPro provides an upgrade path for the Agent Relay. If you are using an older version (i.e., 3.6.2 or older), you will need to install version 3.6.3 in order to perform future upgrades.

*When using SSL, you need to turn it on for both the **server and the agent** -- otherwise the agents will not be able to connect to the server. If you are using the agent relay, ensure that **SSL is turned on for all three components: server, agent, and agent relay**. This rule also applies if using mutual authentication.*

1. Before performing the upgrade, stop the Agent Relay and backup the Agent Relay directory.
2. Run the install script in the Agent Relay installation package.
3. When asked for the directory, give the directory the old version is installed in.
4. The installer will ask you if you want to upgrade. Answer **Y**.
5. Follow the instructions for an installation.

See also Upgrade Distributed Servers and Installing Distributed Servers.

Chapter 21. What's New in AnthillPro 5.0

Welcome to AnthillPro 5.0. Included in the 5.0 release is the introduction of labels, a new feature that makes modeling your SCM branching and dependencies in AnthillPro easier. As with other major releases, version 5.0 includes a host of new improvements that expand AnthillPro's functionality and many improvements.

New Features

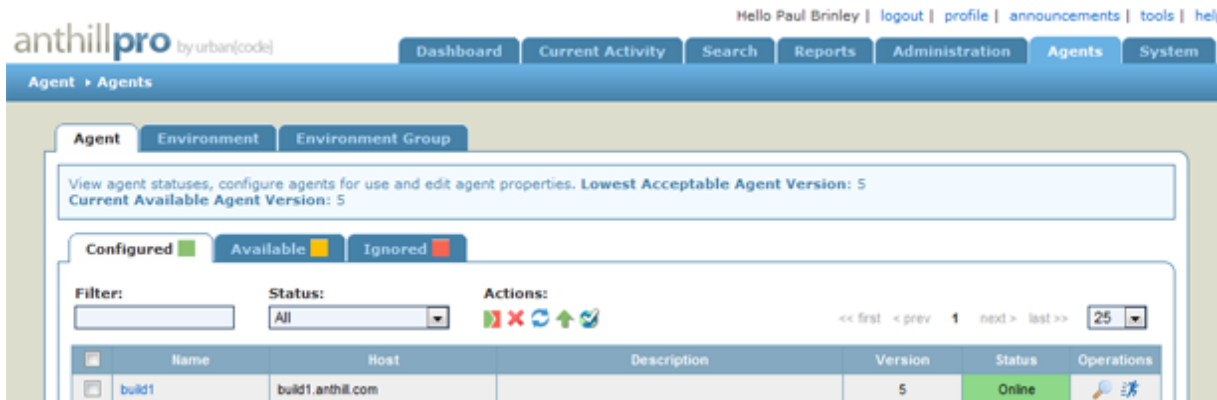
- **Labels.** AnthillPro 5.0 introduces the concept of labels to support branching and creating patch builds. Using the new feature, you can label a Build Life and then use the label when resolving dependencies. Labels can be manually added to a Build Life on the Dashboard page, via the new dependency viewer, or by including an Assign Label step in the build job.
- **Change a project's source.** It is now possible to change an existing project's Source Type directly from the project page. If you select this option, you will have to reconfigure the source for every workflow, in addition to reconfiguring some job steps. See [Changing a Project's Source](#).
- **Current Activity filtering.** Filtering has been added that allows you to save searches.
- **Dependency viewer.** A new way to visualize and configure dependencies has been introduced. A new tab has been added to the Administration page. There, you can configure dependencies, configure new ones, and edit existing dependencies -- all in one place. So, if you have to change a lot of dependency relationships at once, you no longer have to do it from each individual workflow.
- **Request Plans.** Request Plans are a way to save the configuration for running one or more originating workflows. This allows you to run select builds as part of the same request context. See [Use Workflow Request Plan](#).

New Integrations

- **Quality Center.** The integration with Quality Center has been rewritten as an AnthillPro Plugin, providing expanded support over the original Quality Center integration. If you are using the older integration, you can continue to do so. The Quality Center Plugin includes support for custom parameters, and the ability to update Quality Center issues (in addition to the resolve task). See [Quality Center Plugin \(Testing\)](#) and/or [Quality Center Plugin \(Issue Tracking\)](#).
- **Rally.** The integration with Rally has been rewritten as an AnthillPro Plugin, providing expanded support over the original Rally integration. If you are using the older integration, you can continue to do so. The Rally Plugin includes support for multiple Rally workspaces; the ability to have AnthillPro comment on user stories and tasks; update defects and tasks; publish reports for defects, user stories, and tasks; and report the build status back to Rally. See [Rally Plugin](#).
- **Rational Team Concert (SCM).** You can now add any projects in your RTC repository to your AnthillPro build process. See [Rational Team Concert \(SCM\)](#) for configuration instruction.
- **TestNG.** Expanding on the existing functionality, AnthillPro now includes an integration that publishes the TestNG report to the Build Life Test tab, allowing you to track test results over time and drill down into findings. See [TestNG](#).

Improvements

- **Agent and Environment Management UI.** Agent and environment management (including configuration) has been moved to its own top-level tab. This change enables access to agents without having to go to the System page. If you are upgrading from an older version of AnthillPro, you will not need to change any of your existing permission/security settings -- anyone who had access to agents, environments, and/or environment groups will still have them.



For those with a large number of agents, you can use the **Status** field to make searching for agents easier. Once the search is complete, you can then perform an Action on the returned list by selecting the appropriate agent(s).

- **Artifact set UI improvements.** Now, you have a drop-down from which to select the artifact set you want to see the artifacts for. This improves loading time.
- **Cleanup configuration.** Cleanup schedules are no longer configured as part of individual Life-Cycle Models. Instead, a new link has been added to the System page (go to **System > Project Support > Cleanup**). This will enable the same cleanup schedule to be used by multiple Life-cycle Models and Operational Projects. The changes also introduce cleanup of unused working directories, lockable resources, and operational workflows/jobs. See Cleanup.
- **ClearCase UCM Snapshot integration.** It is now possible to define the view strategy, host, gpath and hpath options during source configuration. See ClearCase UCM Snapshot View Source Configuration.
- **Delayed builds and deployments.** Once you configure a delayed build or deployment, it now appears on the **Current Activity** tab under the **Delayed Builds** (or **Planned Workflow Execution** in some versions of AnthillPro) menu. There, you can delete the request for a delayed build/deployment.

Delayed Builds

Project	Workflow	Date	Created By	Log	Details	Actions
Cvs Anthill-Example	Trunk Build Workflow	1/1/11 1:01 AM	admin		12	

- **Drag-and-drop for job steps, authentication realms, and statuses.** Reordering job steps, authentication realms, and Life-Cycle Model statuses is now performed using the drag-and-drop feature. For existing jobs (or when copying steps), authentication realms, and statuses you can reorder them by selecting the grab tool and dragging the step, realm, or status to the new location.
- **Plugin support for global repository triggers.** This allows you set up one trigger in your SCM and have AnthillPro determine which projects need to build based on the information passed along with the trigger request. See the **Developer Kit** (follow the tools link in the upper right-hand corner of the AnthillPro UI) for more details.
- **Preflight builds.** The UI has been redesigned and new functionality added. See the **Developer Kit** (follow the tools link in the upper right-hand corner of the AnthillPro UI) for more details.

- **Property description fields.** You can give a brief description to your environment, project environment, and project properties.
- **Subversion repository trigger and multiple branches.** You can now build multiple branches of source within the same workflow, by setting the Start Stamp Pattern on the SVN Get Changelog step in your build job. See Building Multiple Branches of Source with the SVN Repository Trigger.
- **Working directory script page.** The page has been redesigned to improve loading. Now, the Used In field is a link, and not populated automatically.

Chapter 22. Uninstalling AnthillPro

Both the AnthillPro server and agent are installed within a root directory, so under most circumstances simply deleting them from the file system is sufficient. The only exception is if the server or agent is installed as a service -- in this case it is necessary to remove the service (see here for the server and here for the agent) prior to deleting the server or agent from the file system.

Uninstall the Server

If the server is **not** installed as a service, uninstall AnthillPro by deleting it from the file system after stopping the server. If the agent is installed as a Windows Service, you will need to remove the service before removing the agent from the file system (see below).

To uninstall the server:

1. Shutdown the server.
2. Delete the server from the file system.

To uninstall the server running as a Windows Service:

1. Shutdown the server and stop the service.
2. **Open the Administrative Command Prompt.** Note that you will not be able to remove the service unless you are running as the Windows administrator. For example, run:

```
runas /noprofile /user:mymachine\administrator cmd
```

and then enter the password when prompted. Please consult the Windows documentation if you are having trouble running as the administrator.

3. Go to %SERVER_HOME%\bin\service on the file system where the server is installed.
4. With the Administrative Command Prompt pointing to the server's \bin\service directory, run:

```
ah3server remove <<uniquename>>
```

Ensure that you replace the value **uniquename** with the actual name used to install the server as a service.

5. Verify that the service has been removed.
6. Once the service has been removed, you can delete the server from the file system.

Once the server is deleted, it is also a good idea to uninstall all the agents that were connected to it (see Uninstall the Agent).

Uninstall the Agent

Uninstalling the agent is similar to that of the server -- it simply needs to be removed from the file system. However, if the agent is installed as a Windows Service, you will need to remove the service before removing the agent from the file system (see below).

To uninstall an agent:

1. Shutdown the agent.
2. Remove all references to the agent from the AnthillPro UI. For example:
 - Fixed Agent Filters
 - Fixed Agents on Quiet Period
 - Any Build Lives that reference the agent
 - Any scripts of properties that reference the agent

If you do not remove every reference, any build, etc., that references the removed agent will most likely fail.

3. Ignore the agent in the UI.
4. Delete the ignored agent.
5. Delete the agent from the file system.

To uninstall an agent running as a Windows Service:

1. Shutdown the agent and stop the service.
2. Remove all references to the agent from the AnthillPro UI. For example:
 - Fixed Agent Filters
 - Fixed Agents on Quiet Period
 - Any Build Lives that reference the agent
 - Any scripts of properties that reference the agent

If you do not remove every reference, any build, etc., that references the removed agent will most likely fail.

3. Ignore the agent in the UI.
4. Delete the ignored agent.
5. **Open the Administrative Command Prompt.** Note that you will not be able to remove the service unless you are running as the Windows administrator. For example, run:

```
runas /noprofile /user:mymachine\administrator cmd
```

and then enter the password when prompted. Please consult the Windows documentation if you are having trouble running as the administrator.

6. Go to %AGENT_HOME%\bin\service on the file system where the agent is installed.
7. With the Administrative Command Prompt pointing to the agent's \bin\service directory, run:

```
ah3agent remove <<uniquename>>
```

Ensure that you replace the value **uniquename** with the actual name used to install the agent as a service.

8. Verify that the service has been removed.
9. Once the service has been removed, you can delete the agent from the file system.

Part IV. Schedules

AnthillPro uses schedules to determine when events such as builds, cleanups, backups, etc., are automatically run by the system. Typically, schedules are created on an as-needed basis when you are configuring projects. Once a schedule has been created, it may be used by many different AnthillPro resources. For example, if you want to schedule a nightly build for a project, set up a schedule that fires once a day and then add that schedule to the project's trigger. Or, you can set up a cleanup schedule that is used by numerous projects. There are two types of schedules you can create in AnthillPro:

- **Interval Schedule.** Regularly fires after a fixed interval of time, and is the simplest schedule. See [Create Interval Schedule](#).
- **Cron Schedules.** May be configured to consider more complex criteria than a simple interval. See [Create Cron Schedules](#).

Which schedule type you use is based, in part, on the complexity of the event you are scheduling. For daily/nightly builds, the interval schedule is sufficient. However, for backups, you may want the schedule to fire the first Saturday of the month -- except on holiday weekends, etc. In this case use a Cron schedule.

To create a schedule, you must have permissions to the System page. See [Setting Up Security](#).

Chapter 23. Create Interval Schedule

The **Interval Schedule** regularly fires after a fixed interval of time, and is the simplest schedule. These are typically used for nightly builds, backups, cleanups, etc.

To create a Schedule, you must have Administrative permissions. See Setting Up Security.

1. Go to **System > Schedules** under the Project Support menu.
2. Click the **Create Schedule** button.
3. Check the schedule type:
 - **Interval Schedule.** Fires after a fixed interval of time passes.
4. Click **Set**.
5. Configure Interval Schedule:
 - **Name** the schedule.
 - **Description.** Give a description of the schedule.
 - **Build Interval (minutes).** Give the amount of minutes that will pass before the scheduled task runs.
 - **Interval Start Time (hh:mm).** Provide the time at which this schedule should start running. Note that the start time will be determined by the time zone that the AnthillPro server is in.
6. Click **Set** then **Done**.

Chapter 24. Create Cron Schedule

Cron Schedules may be configured to consider nightly or weekly outages of servers (for backups), weekly builds, or builds that occur on every other Friday except in February, etc.

To create a Schedule, you must have Administrative permissions. See Setting Up Security.

1. Go to **System > Schedules** under the Project Support menu.
2. Click the **Create Schedule** button.
3. Check the schedule type:
 - **Cron Expression Schedule.** Fires based on a Cron expression.
4. Click **Set**.
5. Configure Cron Schedule.
 - **Name** the schedule.
 - **Description.** Give a description of the schedule.
 - **Cron Expression.** Provide the Cron expression to use for determining the schedule. *See the **New Schedule** page of the AnthillPro UI for detailed instructions.*
6. Click **Set** then **Done**.

Part V. Reporting

AnthillPro ships with a number of built-in reports that provide information about system activity. Available on the main **Reports** page, you can easily run a report that:

- Displays builds by day of the week.
- Shows all the failed and successful builds by time of day.
- Lists all the Codestation Artifacts.
- Provides a list of all projects and their immediate dependencies.
- Returns a detailed list of all projects and their direct dependencies.
- Gives the size of all artifact sets associated with a project.
- Lists recent Build Life Activity (includes support for RSS).

If the standard reports do not meet your needs, AnthillPro provides a flexible utility to create custom, tabular reports and RSS feeds. There are no restrictions on what can be reported on.

Custom reports are generated by using scripts that have the full AnthillPro database and API at their disposal. Reports can be designed to produce well-formatted output like XML files, and can be accessed using simple HTTP requests -- making them a great integration point. You can design reports to produce information of interest that other applications can look up.

You can create two types of reports:

- **Velocity Reports** use a Velocity template unique to each report to render the content. Velocity Reports are good for customizing the AnthillPro UI. When integrated, an Apache Velocity [<http://velocity.apache.org/>] report will be passed to the Build Life page, where they can be directly accessed. Once created, Velocity Reports can be run directly from the Build Life Reports tab. See [Creating a Velocity Report](#).

You must have administrative permissions to the System page to create custom reports (see [Setting Up Security](#)).

- **Template Reports** are good for generating tabular data in one or more types of output. AnthillPro ships with support for HTML, CSV, RSS, and Bar Chart formats, which should meet most of your needs. However, you can write your own report template to support a different format type. See [Creating a Template Report](#).

All reports -- including your custom reports -- may be made public (go to **System > Reports**, select the report, and activate the **Publicly-Available** option). This will make the report URL available to people who do not have AnthillPro login permissions (e.g., managers, executives, etc.).

Chapter 25. Running a Report

Once a report has been created (see [Creating a Template Report](#) and [Creating a Velocity Report](#)), the new report becomes available on the **Reports** page, along with the standard AnthillPro reports. To run a report:

1. Go to **Reports** and select the **Run Report** icon for the report you want to run.
2. Select the **Report Template** and click the **Run** button to generate the report.

Ensure that the correct Report Template is selected. The desired output determines which template to use. For example, the output of the Builds by Time of Day report supports the Bar Chart option; however, it does not support RSS. If a "report output does not support the template" exception is thrown, select a different template from the drop-down menu. To add a custom template type, see [Writing a Report Template](#).

3. When the report opens in the browser, you can bookmark the URL for future viewing.

It is possible to make individual reports publicly available to those without AnthillPro login credentials. To do so, simply check the **Publicly Available** option on the report configuration page (see [Creating a Template Report](#) and [Creating a Velocity Report](#)).

Chapter 26. Creating a Template Report

Template Reports are good for generating tabular data in output such as HTML, CSV, and RSS -- all supported by AnthillPro. To generate a report, you will need to write a report script that returns the content of the report. In addition, you will also need to make sure the content matches one of the report-template types, otherwise the contents may not display properly. Because column names and numbers are determined by the report scripts, a single report template may be used by many report scripts.

Detailed instructions, including examples, on writing Report Scripts is available. To view the documentation, go to **Tools** and download the **Development Kit Bundle**. In the **scripting** directory, open **Scripting.pdf** and scroll down to the **Reporting Scripts** section. Urbancode also maintains a publicly available list of report and report-template scripts that may be helpful when writing a custom report. They are viewable here [<https://bugs.urbancode.com/secure/IssueNavigator.jspx?reset=true&mode=hide&pid=10110&component=10147>].

If one of the standard template types does not meet your needs, AnthillPro allows you write your own Report Template.

1. Go to **System > Reports** under the Reporting menu. Make sure you have administrative permissions to the System page (see Setting Up Security).
2. Click the **Create Report** button on the **Reports** main page.
3. Check the **Template Report** box and click **Select**.
4. **Name** and give a **description (optional)** of the report.
5. **Publicly Available**. Check the box to make the report URL available to people who do not have AnthillPro login permissions (e.g., a manager, executive, etc).
6. **Meta-Data Script**. For the most part, this script is just responsible for listing the columns that will be used by the report. Enter a BeanShell script that should return a `ReportMetaData` object which describes the inputs to the Report Script below, as well as the data fields of the report itself.
7. **Report Script**. Enter a BeanShell script that creates the report and returns a `ReportOutput` object. This script will receive the inputs that were defined as parameters in the Meta-Data Script above. All the inputs will be passed to the script as parameters with their respective names. The `ReportMetaData` object will also be supplied to the script as the parameter "metaData". The key thing is to create a `ReportOutput` object using the meta-data created in the script above and add `ReportRows` to it.
8. Click **Save**.

Chapter 27. Creating a Velocity Report

Velocity Reports use a Velocity template unique to each report to deliver the content. Velocity Reports are good for customizing the UI, as they can automatically be made available on the Build Life page. Once created, Velocity Reports can be run directly from the Build Life Reports tab.

Detailed instructions, including examples, on writing Report Scripts is available. To view the documentation, go to **Tools** and download the **Development Kit Bundle**. In the **scripting** directory, open **Scripting.pdf** and scroll down to the **Reporting Scripts** section. Urbancode also maintains a publicly available list of report and report-template scripts that may be helpful when writing a custom report. They are viewable here [<https://bugs.urbancode.com/secure/IssueNavigator.jspx?reset=true&mode=hide&pid=10110&component=10147>].

For advanced users, the Apache Velocity Project [<http://velocity.apache.org/>] provides numerous resources that may be of benefit.

1. Go to **System > Reports** under the Reporting menu. Make sure you have administrative permissions to the System page (see Setting Up Security).
2. Click the **Create Report** button on the **Reports** main page.
3. Check the **Velocity Report** box and click **Select**.
4. Configure report:
 - **Name** the report.
 - **Description**. Provide a description.
 - **Content Type**. Optionally, you can give the MIME type of the report content. This will be sent to the browser when the report content is generated. E.g.: text/plain, text/csv, or text/html.
 - **Publicly Available**. Check the box to make the report URL available without having to login to AnthillPro. This is helpful for managers, executives, etc., who need to view the report.
 - **Integrations**. Check the box to select the UI integration points for this report. When integrated, the report will be available directly in the UI and the context script will be passed to the Build Life as 'buildLife'.
 - **Context Script**. Give an optional BeanShell script to populate the context of the template. The script should return a Map object as the context for the Velocity template to render. This script will be passed all request properties when it is run, as well as a Map object called 'context'.
 - **Template Text**. Input the BeanShell script that creates the report and returns a `ReportOutput` object. This script will receive the inputs that were defined as parameters. All the inputs will be passed to the script as parameters with their respective names. The `ReportMetaData` object will also be supplied to the script as the parameter "metaData".
5. Click **Save**.

Chapter 28. Writing a Report Template

A key role of the report template is to manage the format of the data being displayed. The `ReportOutput` object (see [Creating a Template Report](#) and [Creating a Velocity Report](#)) is passed into the template as output and the meta-data is passed in as meta-data.

AnthillPro ships with 4 standard Report Templates that should meet most of your needs:

- **Bar Chart.** Displays the data in a bar chart embedded in HTML. Data must all be numeric.
- **CSV.** Renders a comma-separated CSV format.
- **HTML (Anthill Style).** Renders HTML output in a table form using the Anthill style.
- **RSS 2.0.** Renders a RSS feed.

If you need to render the report in a different format, follow the instructions below for creating a new template type. Once that is done, the new template type will be available as an option to anyone who runs a report.

Detailed instructions, including examples, on writing Report Scripts is available. To view the documentation, go to **Tools** and download the **Development Kit Bundle**. In the **scripting** directory, open **Scripting.pdf** and scroll down to the **Reporting Scripts** section. Urbancode also maintains a publicly available list of report and report-template scripts that may be helpful when writing a custom template. They are viewable here [<https://bugs.urbancode.com/secure/IssueNavigator.jspx?reset=true&mode=hide&pid=10110&component=10147>].

1. Go to **System > Report Template** under the Reporting menu. Make sure you have administrative permissions to the System page (see [Setting Up Security](#)).
2. Click the **Create Template** button.
3. **Name** the new template, provide a **description (optional)**, and provide the following:
 - **Content Type.** Give the MIME type of the report content. This will be sent to the browser when the report content is generated. E.g.: `text/plain`, `text/csv`, or `text/html`.
 - **Context Script.** Optionally, give the context script. For many report templates, the context-script can be left blank.
 - **Template Text.** Enter the content of the template here. A `ReportOutput` object will be automatically provided to the script as "output". The Report object will also be provided to the script as "report". For an example, see below.
4. Click **Save**.

Example template for a review of Build Life events

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Workflow</title>
<STYLE TYPE="text/css">
```

```

<!--
table.data-table td {
    vertical-align: top;
}

table.data-table
{
    font-family: arial, helvetica, sans-serif;
    font-size: 12px;
    background-color: #567596;
}

table.data-table caption
{
    padding-top: 10px;
    padding-bottom: 10px;
    text-align: left;
}

table.data-table th
{
    text-align: center;
    background-color: #cfdbef;
    height: 25px;
}

table.data-table td
{
    vertical-align: top;
}

table.data-table tr.odd
{
    background-color: #ffffff;
}

table.data-table tr.even
{
    background-color: #f6f6f6;
}

.data-table-button-bar
{
    padding-top: 10px;
    padding-bottom: 10px;
}

.data-table-container
{
    padding-top: 10px;
    padding-bottom: 10px;
}
-->
</STYLE>
</head>
<body>

<h1> Report: $report.Name</h1>
<p>
<div class="data-table-container">
<table class="data-table" cellpadding="4" cellspacing="1" width="100%">
  <table-body>
    <tr class="data-table-head">

```

```
#foreach($column in $output.MetaData.ColumnArray)
  <th scope="col" align="left" valign="middle"><strong>$column</strong></th>
#end
</tr>
#foreach($row in $output.RowArray)
  <tr bgcolor="#ffffff">
    #foreach($columnValue in $row.ColumnValueArray)
      <td>$columnValue </td>
    #end
  </tr>
#end
</table-body>
</table>

</body>
</html>
```

Chapter 29. RSS and Reporting

A report designed for RSS will need RSS-style rows. RSS works nicely with the report system since requests for reports are all done using HTTP GETS. The generated URLs will work for an automated connection. A context script that checks for the presence of those rows can be helpful.

Typical context script for an RSS (2.0) feed

```
import com.urbancode.anthill3.domain.reporting.*; ReportOutput output =
    (ReportOutput)
context.get("output"); // validate that the report can be turned into a
    RSS feed if
(!output.hasColumn("title") || !output.hasColumn("date") ||
 !output.hasColumn("link") ||
!output.hasColumn("guid") || !output.hasColumn("description"))
    { throw new Exception
      ("The report output
does not support the RSS template"); } context.put("pubDate", new Date());
```

With the context script done, you then pair it with the RSS template (similar to what is below):

RSS template paired with context script

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>${report.Name}</title>
    <link>${reportUrl}</link>
    <description>${report.Description}</description>
    <language>en-us</language>
    <pubDate>${pubDate}</pubDate>
    <generator>Anthill3</generator>
    #foreach($row in $output.RowArray)
      <item>
        <title>${row.getColumnValue("title")}</title>
        <link>${row.getColumnValue("link")}</link>
        <description>${row.getColumnValue("description")}</description>
        <pubDate>${row.getColumnValue("date")}</pubDate>
        <guid>${row.getColumnValue("guid")}</guid>
      </item>
    #end
  </channel>
</rss>
```

Part VI. Dependency Management

Often, an application will require outside libraries or other files to build: the dependent (parent) project will use the artifacts (e.g., versioned, released binaries) produced by a child project. This guideline is easy to follow if the child's artifacts never, or rarely, change. In this case, you can check the project into source control and manually update the library when you decide to move to a different version. However, the parent project can also rely on the artifacts generated by other projects your organization develops. When this is the case, manually updating dependencies becomes increasingly difficult, and a more robust dependency-management solution is needed.

To address these needs, AnthillPro includes a dependency-management system -- called Codestation -- that provides fine-grained management for your dependencies. Through the AnthillPro UI, you are able to configure and manage dependencies for:

- **Third-party libraries and tools.** Most organizations will use externally developed libraries, projects, etc. For example, the parent project may depend on a specific version of the Apache Tomcat server that your organization does not actively develop. AnthillPro provides a special project type -- called Codestation Projects -- to manage these dependency types. A Codestation project is always at the bottom of the dependency tree: it can never depend on another project (i.e., it must always be a child project).
- **Internal projects.** Allows for dependency relationships between concurrently developed AnthillPro projects. For example, your application may be composed of numerous components or modules that are independently built. For these types of projects, dependencies are configured in much the same way as with Codestation dependencies. The main difference is that for internal dependencies, the parent project may also be a child project (e.g., project A depends on project B, which in turns depends on project C; however, cycles are not allowed). AnthillPro is configurable to automatically update a parent project with the latest releases of the projects it depends on; lock down the dependency on a specific version (build) of the artifacts; and control whether a dependency should kick off a build or not.

In AnthillPro, dependencies are configured at the workflow level (Codestation projects are a bit different; more on that later), and depend on the artifacts associated with the Build Life of the child project. For example, if your parent project depends on the artifact sets of 2 other project Build Lives, you configure 2 dependencies on the *parent's* build (originating) workflow -- one for each project (see Configure Dependencies and Using Codestation Projects).

When configuring dependencies, you will need to:

1. Determine which project your parent project depends on. This information will usually come from your existing dependency-management system or build scripts.
2. Configure an artifact set on the project your parent project depends on (i.e., the child project). This should include all the files that your parent project needs to successfully build. A publish artifacts step will also need to be added to the child project. See Capture and Deliver Build Artifacts (Child Project).

If the child project is a Codestation project, see Using Codestation Projects.

3. Go to your parent project and add a new dependency configuration. You will need to complete the configuration on the **Dependencies** tab of the originating (build) workflow, as well as add a resolve dependencies step to the build job. See Configure Dependencies.

Once the system is configured, AnthillPro maintains a record of which Build Life the artifacts came from. This provides instant traceability from the parent project back to its own source files, as well as those of each of its dependencies (the children projects).

AnthillPro also allows you to use file-defined dependency configurations. You can use a Codestation method to set

the dependencies of a Build Life from a file-defined resolve and then add it to the client and CLI. File-defined dependencies, using an XML file, is supported for both historical resolves and transitive dependencies. Once your file is configured, you can have Codestation pull down the dependencies. For more, go to **Tools > Developer Tools > Codestation Client**.

Chapter 30. Configure Dependencies

Dependencies are configured on originating (build) workflows so that the artifacts of the child project can be retrieved at run time. Codestation, AnthillPro's dependency-management system, enables you to set how dependency builds occur. For example, a dependency can be configured so that the parent project checks its dependencies and builds them at run time. This is called a pull build. Or, a build for the child project can kick off a build of the parent. This is called a push build.

To successfully configure a dependency relationship requires configuration on both the parent and child projects:

1. **Artifact Set configuration on the child project.** AnthillPro stores the build artifacts of the child project in Codestation as an Artifact Sets. See [Capture and Deliver Build Artifacts \(Child Project\)](#).
2. **Dependency configuration on the parent project.** Once the child project's build artifacts are captured and delivered to Codestation, the dependency is configured on the parent project's originating workflow. You will also need to add a Resolve Dependency Artifacts step to the parent project's build job. See [Configure Dependency \(Parent\)](#).

Dependencies may also be configured using the Dependency Viewer on the Administration tab. While the view is different, the basic process is the same as outlined below. The one main advantage to using the Dependency Viewer is that you can manipulate multiple dependency configurations without having to navigate to the individual projects.

Dependency Configuration Prerequisites

- You will need read and write permission to the Administration page, as well as administrative permissions for configuring Life-Cycle Models if you need to add a new Artifact Set. See [Setting Up Security](#) or contact your AnthillPro administrator.
- The child project should already have at least one successful Build Life. If you are configuring a dependency on a Codestation project, make sure you have configured a Build Life. See [Using Codestation Projects](#).

Capture and Deliver Build Artifacts (Child Project)

Before you can complete a dependency relationship, AnthillPro will need to capture the child project's build artifacts and then deliver those artifacts to the embedded dependency-management system (called Codestation). Once the artifacts are in Codestation, AnthillPro will be able to keep track of when and where they are used as dependencies.

Since AnthillPro does not know what artifacts you want captured, you will need to label them so they can be grouped and stored together. This is done by defining an artifact set on your build process. You can think of the artifact set as a grouping of build artifacts (files) that allows for fine-grained consumption of the artifacts. Once the artifacts are labeled and grouped into an artifact set, you will need to manually add a Deliver Artifact step to your build job.

To capture and deliver build artifacts:

1. Go to **Administration** and select the child project's build workflow.

2. On the Workflow page, select the **Artifacts** tab, then click the **New Artifact Config** button.
3. Configure artifact set:

- **Artifact Set.** Select an artifact set from the drop-down menu (using one of the defaults is fine if you are just starting out). AnthillPro ships with 3 default artifacts sets that correspond to the most common tiers of a 3-tier application:
 - **APP.** Used to group objects for deployment to the application tier of a 3-tier application.
 - **DB.** Used to group objects for deployment to the database tier of a 3-tier application.
 - **WEB.** Used to group objects for deployment to the web tier of a 3-tier application.

Generally, an artifact set is named for the type of objects which are grouped inside of them. The objects within an artifact set, in turn, are grouped by how the objects are going to be consumed.

Most users will eventually need to configure their own artifact sets to better reflect the contents. The process is rather simple for an existing project: You simply add a new artifact set to the project's Life-Cycle Model.

- **Base Directory.** Here, you need to give the directory where the artifacts (say a jar or dll file) are placed once the build is complete. This directory is relative to the build's working directory. So if your build places the artifacts in a "dist" directory, you would specify `dist/` here. Note that if you leave this field blank, AnthillPro will include the entire contents of the working directory in the artifact set.
- **Include Artifacts.** List the artifacts to be retrieved from within the base directory. You can specify the names of files that reside in the base directory: e.g., `myProjectArtifacts.zip`. Or, if the artifacts are located in a sub directory, you specify something like `bin/myProjectArtifacts.jar`. Each include pattern must be entered on a separate line.

You can also use the following wild cards to tell AnthillPro what to include in the artifact set:

- `**` Indicates include every directory within the base directory.
- `*` Used to include every file. So, if you use `*.zip`, the files matching this pattern will be included.
- `**/*` Tells AnthillPro to retrieve the entire file tree underneath the base directory.

If you leave this field blank, AnthillPro will include all the files in the base directory (if one was specified).

Advanced: To include symbolic links and empty directories, add the link/directory as part of your include pattern. See Configure Server Miscellaneous Settings.

- **Exclude Artifacts.** Give the patterns of the artifacts that should be skipped from the include. This field is set in the same way as the Include Artifacts field, only you are telling AnthillPro what NOT to include. If you leave this field blank, AnthillPro will exclude no files.
- Click **Save**.

4. To deliver your build artifacts to Codestation, you need to add the **Artifact Delivery** step to the build job. Go to the child project's job configuration page (on the **Administration** page).

In order for AnthillPro to manage the artifacts, they must be sent to Codestation, AnthillPro's built-in artifact/dependency management system. If the deliver step is not included, the artifacts will remain in the working directory (until it is cleaned up, etc.) and unavailable for dependency relationships.

5. Click the **Insert After** icon of the step prior to where this step should run. Remember, the steps run in order, so any step, such as the Artifact Deliver step, which deals with the build artifacts must come after the actual "build"

step.

6. On the Steps page, expand the **Artifacts** folder, select **Artifact Deliver**, and click **Select**.

7. Configure step:

- **Name.** Give a name for this step. A simple "Artifact Deliver" is sufficient.
- **Artifact Set.** Select the artifact set you just finished configuring.
- **Show Additional Options.** These are advanced settings. For a simple dependency relationship, you can skip these settings.
- Click **Save**.

8. **Run a build of the child project.** The artifacts will be delivered to the AnthillPro dependency-management system. This will make the artifacts available as a dependency to other projects.

To view the artifacts, go to the Dashboard and select your build workflow. Click on the most recent **Build Life** (i.e., build number) and then select the **Artifacts** tab.

9. After verifying the correct artifacts have been captured and delivered, see Configure Dependency (Parent).

Configure Dependency (Parent)

Once the child project's artifacts are in Codestation, you need to configure the dependency relationship on the parent project's originating (build) workflow, and add a resolve dependency step to the build job. When configuring dependencies, you will also need to tell AnthillPro how you want your dependencies handled. To get started:

1. Go to **Administration** and select the parent project's **originating (build) workflow**.

2. Select the **Dependencies** tab.

3. Click **New Dependency** at the bottom of the page. Configure the dependency:

- **Project.** Begin typing the name of the child project. You can use the wild card * (star) to return all available projects. The auto fill will give you both the name of the project and associated workflows. If the child project has multiple originating workflows, make sure you select the correct one. See Capture and Deliver Build Artifacts (Child Project). Click **Select**.
- **Build Life Criteria.** Allows you to define which Build Life on the child project will be used to fulfill the dependency. The criteria for this is based on the status and/or stamp value.

The values given here will also be used when a build, based on the dependency trigger (see below), fails. Should a requested child-project build fail or if one is not needed, AnthillPro will use the **Build Life Criteria** to determine which child-project Build Life to use when building the parent project. Build Life Criteria is cumulative. Having "success" and "1.2.*" will match successful builds whose build number starts with 1.2.* -- not with successful builds or those that start with 1.2 (without the final period).

- *With Status.* Select a status from the drop-down menu. The statuses used here were configured on the child project's Life-Cycle Model. If you select a status, only Build Lives that assigned the status are available to the parent project.
- *With Stamp.* You can either input the exact stamp or specify a stamp pattern using '?' for a single character

and '*' for matching any sequence of characters. If you use a stamp, make sure it matches the stamp used on the child project.

If configuring a dependency on a Codestation project, make sure you have assigned the appropriate Status and Stamps on the that project's configuration page. See Using Codestation Projects.

- **Dependency Trigger.** Determine how AnthillPro should treats builds of the parent project.

Before configuring a Dependency Trigger, please see Pulling Builds and Pushing Builds.

- *None.* AnthillPro will look up the matching build and use it. No additional builds will be created by the dependency relationship.
- *Pull Build.* Before the parent project's workflow runs, it will create a new build request for the child project. See Pulling Builds.

Use existing if pull fails. AnthillPro will use the most recent child-project build if the request results in a failed build.

Always force. Will always build, even if a failure occurs down the dependency graph.

Cascade force. Forces a build of all the projects down the dependency tree.

- *Push Build.* Pushes builds up the dependency relationship. Each time a new build of the child project meets the criteria, a build request for the parent project's workflow is created. See Pushing Builds.
- **Artifact Retrieval.** Tell AnthillPro which **Artifact Set** to put in what location. The locations are relative to the workflow working directory. Check the box next to each artifact to retrieve.
 - *Retrieve.* Check the box next to the appropriate artifact set.
 - *Transitive.* Check to retrieve dependencies of the dependency into the same folder. The grandchildren must have an artifact set of the same name and the dependency must also check the transitive box for each of those dependencies.
 - *Location.* Specify the relative path(s) from the working directory (of the parent project) to copy the dependency artifacts to. You can set multiple locations by using the add link. Project and workflow properties are the only scripting allowed. E.g. `${property:deps_dir}`. Use '.' for the current working directory, leave empty to remove.
- Click **Save**.

4. **Conflict Strategy.** Tell AnthillPro how you want to deal with dependency conflicts.

The conflict strategy works a bit differently for pulling and pushing dependency triggers. For examples, see Dependency Conflict Strategy.

Click the **Edit** button under the Conflict Strategy menu and select one of the following from the drop-down:

- **Fail.** This option will fail the parent-project's build when a conflict is detected. If your notifications are configured correctly, AnthillPro will notify you that the build failed. The parent project will not build until you correct the conflict.
- **Favor old.** Use this option to have the oldest Build Life's artifacts overwrite those from newer Build Lives.

AnthillPro will always use the oldest available artifacts from the child project when a conflict occurs.

- **Favor new.** Allows the newest Build Life's artifacts to overwrite those from older Build Lives. AnthillPro will use the most recent child-project artifacts when a conflict occurs.

5. **Add Resolve Dependency Artifacts job step.** This will allow AnthillPro to retrieve the artifacts generated by the child project.

Go to **Administration** and select the parent-project's **build job**. Click the **Insert After** icon of the step prior to where the resolve step is to be included. This should be added *before* the build step, typically after the populate, changelog, and stamp steps. Configure step:

- **Name.** Give the step a descriptive step. For example, Get Dependency Artifacts.
- **Transfer Only Changed Files.** Retrieves only files that are not already present. To have AnthillPro transfer only changed files based on checksums of the secured artifacts, check the box. Before you run the build, see Secure Artifact Sets.
- **File Include Patterns.** Leave blank or use `**/*` for all artifacts. To include specific files, give the name patterns that describe the files that will be resolved.
- **File Exclude Patterns.** Give the file name patterns identifying the files that will NOT be resolved.
- **Artifact Set Include Patterns.** Give the patterns matching artifact set names that will be resolved. Use `*` to match zero or more characters, or `?` to match exactly one character.
- **Artifact Set Exclude Patterns.** Give the patterns matching artifact set names that will be resolved. Use `*` to match zero or more characters, or `?` to match exactly one character.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

6. Click **Save**.

Pulling Builds

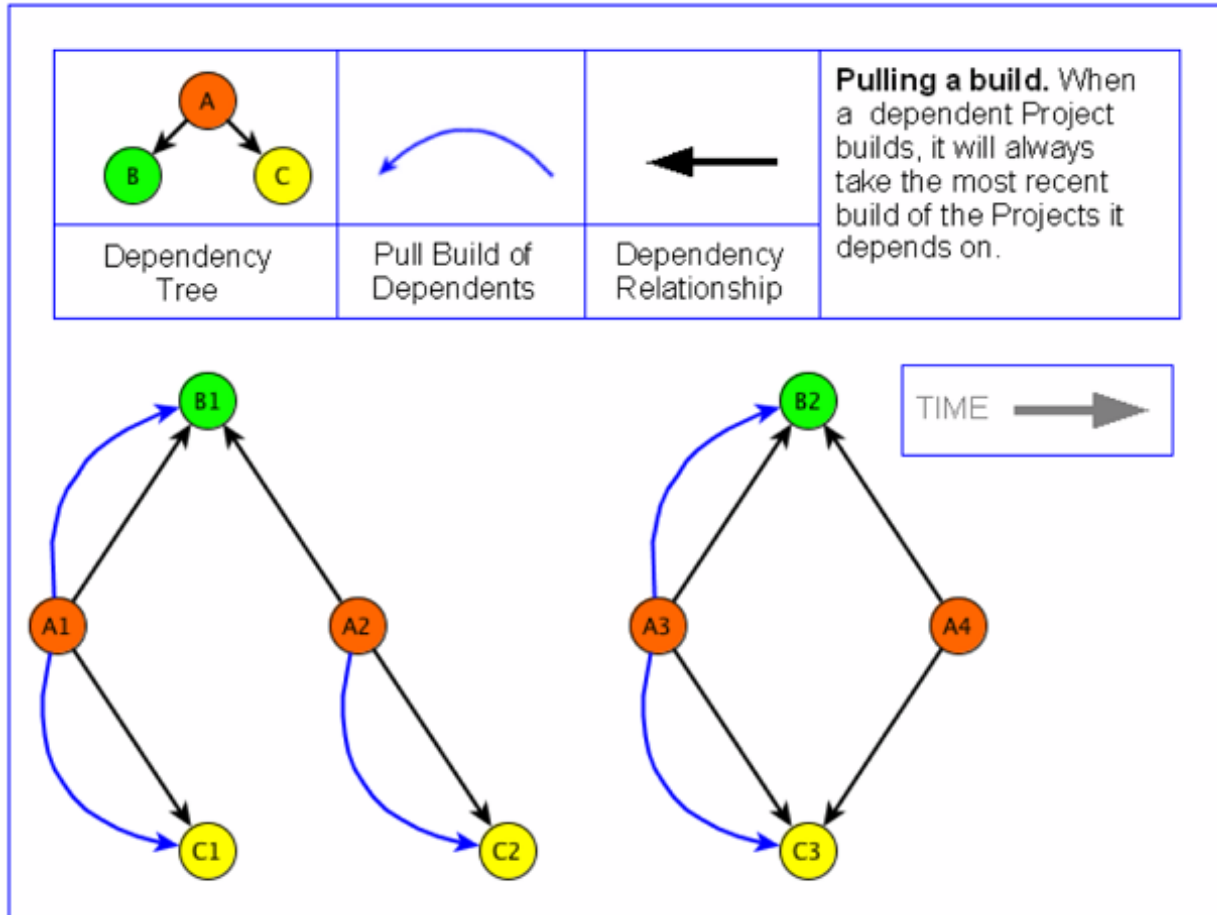
Pull builds will build every dependency that is out of date prior to building the top level project. Before the parent-project's workflow runs, it will create a build request for the child project. The build request may or may not result in a new Build Life of the child project. If the request determines changes have been made to the child project since the last time the parent project was built, the request will kick off a new build of the child project. When the child project finishes building, the new Build Life will be used to fulfill the dependency (see example below).

- If a Build Life is not created or the creation of the Build Life fails, AnthillPro will use the **Build Life Criteria** to determine which child-project Build Life to use when building the parent project. If you are setting a dependency trigger, make sure the Build Life Criteria is configured. Build Life Criteria is cumulative. Having "success" and "1.2.*" will match successful builds whose build number starts with 1.2.* -- not with successful builds or those that start with 1.2 (without the final period).

Pulling Builds

When project A builds, it checks for any new builds of projects B and C since the last time project A was built. For build A1, both projects B and C have been built since the last time project A built, so the project A build kicks off (pulls) builds B1 and C1 and then uses the artifacts of the new builds. For build A2, only project C has changed, so build C2 is pulled and used. Since project B has not changed since the last build of project A, build B1 is used for build A2.

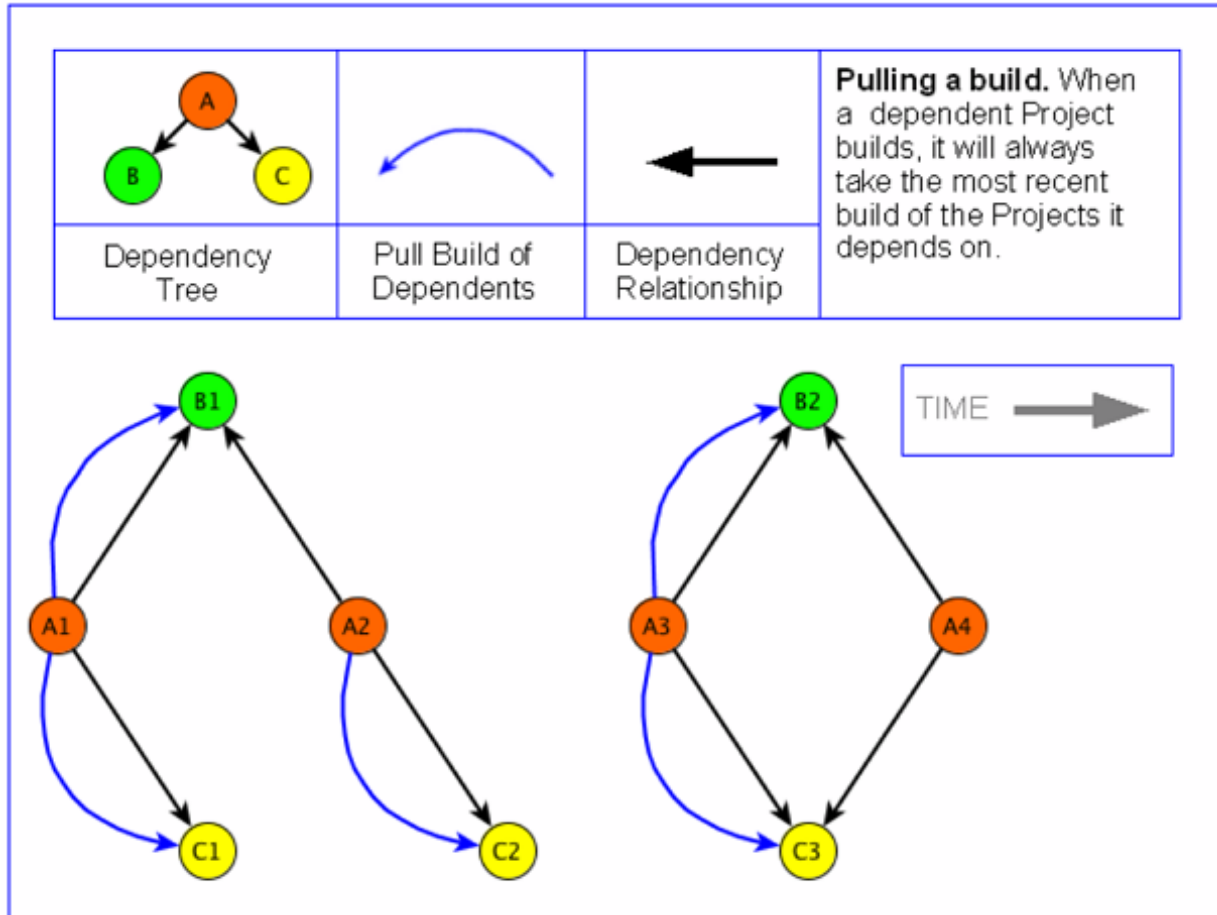
When project A builds for a third time (A3), it detects changes in both dependent projects, so it pulls a build for both projects (B2 and C3 respectively). For build A4 of project A, no builds of the dependent projects (A and B) are required because no changes have been made to either project since the last build of project A (build A3). So, build A4 simply uses the artifacts of builds B2 and C3 during the build.



A Pull Dependency configuration uses less resources than pushing builds, and can be used if resources are scarce. However, there is a trade off when compared to pushing builds: A pulled build does not have the same guarantee that every child project down the dependency tree will compile when the parent project builds. This is highlighted when the dependency trigger is configured to use an existing build if a pulled build fails (see example below).

Use Existing Build if Pull Build Fails

Each project in the dependency tree successfully built (builds B0, C0, and D0 respectively) when the parent project A0 was built. When another build (A1) of project A (the parent project), it created a request for a build of all the projects down the dependency. However, the second build of project C (C1) failed. Because AnthillPro was configured to use an existing build of a child project, the build of project A (the parent) takes the most recent successful build of project C (C0).



If your notifications are configured correctly, AnthillPro will notify you that the build of project C (C1) failed. You can then view the Request Context of project C's failed build to diagnose the problem.

Pushing Builds

Push builds perform the minimum number of builds, in the correct order, to ensure that a change to a component does not break anything that depends on it directly or transitively. In a typical scenario core libraries (represented by project C in the illustration) rebuild rarely and so rarely trigger builds. If low level libraries change frequently, Push builds will rebuild large parts of the tree regularly causing many builds.

It is worth noting that push builds can become resource intensive, especially when you have a large dependency tree. However, pushing builds provides the best assurance the parent project will compile successfully (see example below).

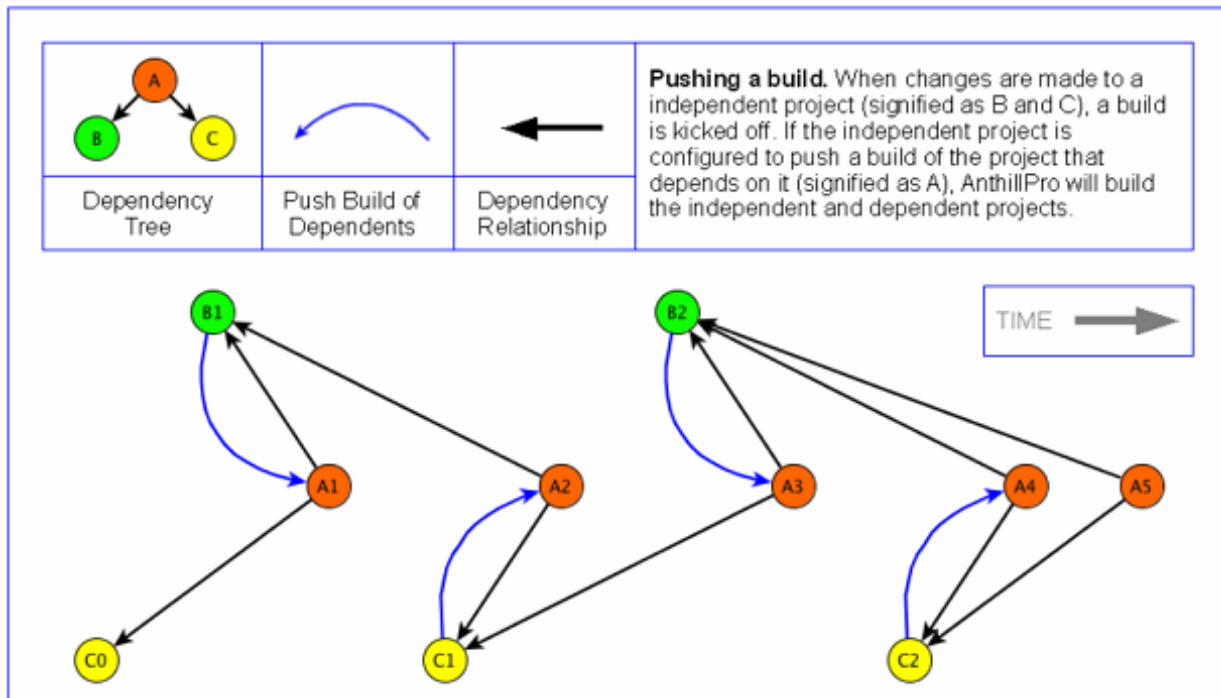
Pushing Builds

When changes are made to a child project (projects B and C), a build is kicked off. If a **push-build dependency trigger** is configured on the parent project (project A), AnthillPro will build both the child and parents.

For example, when project B builds (B1), a build of project A is pushed (kicked off), creating build A1.

When project C builds (C1), a build of project A is pushed. Likewise, when project B builds again (B2), a build of project A is pushed, creating build A3. The pushed build of project A then checks the dependency relationship with project C, and uses the most recent project C build (C1).

When project C builds (C2), a build of project A is pushed, creating build A4. Build A4 uses build B2. So when a build of Project A is kicked off (A5), it will use the most recent builds of projects B and C (B2 and C2 respectively).



Dependency Conflict Strategy

The dependency conflict strategy allows you to control how AnthillPro acts when a conflict occurs within the dependency tree. To configure, see Conflict Strategy.

Conflicts typically show up when the parent project (project A in the examples below) depends on two different child projects (B and C respectively) that depend on the same project (D). Looking at the examples below, the build requests are conflicted because two different builds of the same project have been requested in the same context.

When determining the conflict strategy, you can select the following options:

- **Fail.** This option will fail the parent project's build when a conflict is detected. If your notifications are configured correctly, AnthillPro will notify you that the build failed. The parent project will not build until you correct the conflict.
- **Favor old.** Use this option to have the oldest Build Life's artifacts overwrite those from newer Build Lives. In the

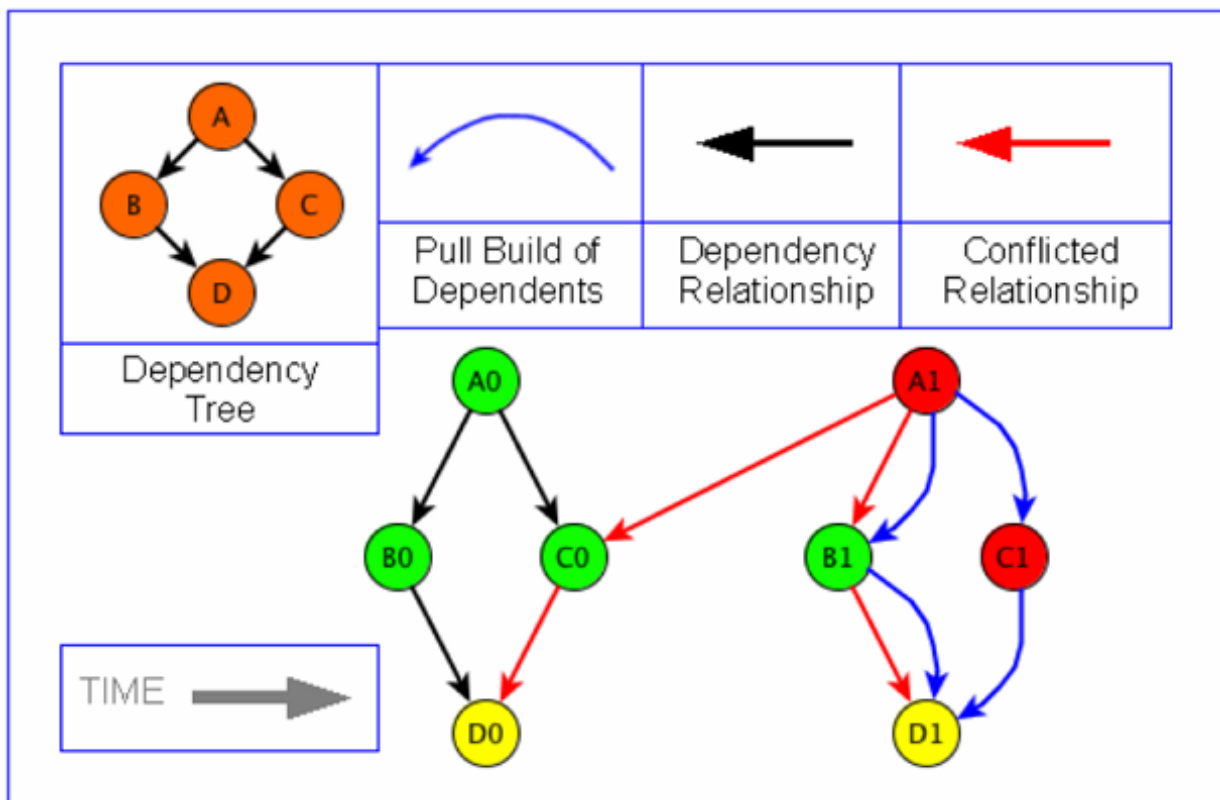
examples below, AnthillPro would use project D's original build (D0) as part of build A1 for the parent project.

- **Favor new.** Allows the newest Build Life's artifacts to overwrite those from older Build Lives. AnthillPro will use the most recent child-project artifacts when a conflict occurs. In the examples below, AnthillPro would use project D's newest build (D1) as part of build A1 for the parent project.

Conflict Strategy with Pull Dependencies

When project A needs to build (A1), it creates a request for build B1 and C1. Likewise, builds B1 and C1 both request a build of project D (i.e., they both want build D1) within the same context. This is where the conflict comes in: Build C1 fails, and build A1 reverts to build C0. Unfortunately, C0 depends on build D0 and build B1 depends on build D1. Because two versions of D are in use, there is a conflict.

The conflict strategy is used to determine which version of project D to use when building project A (build A1) or to fail the build of project A.



See also Pulling Builds.

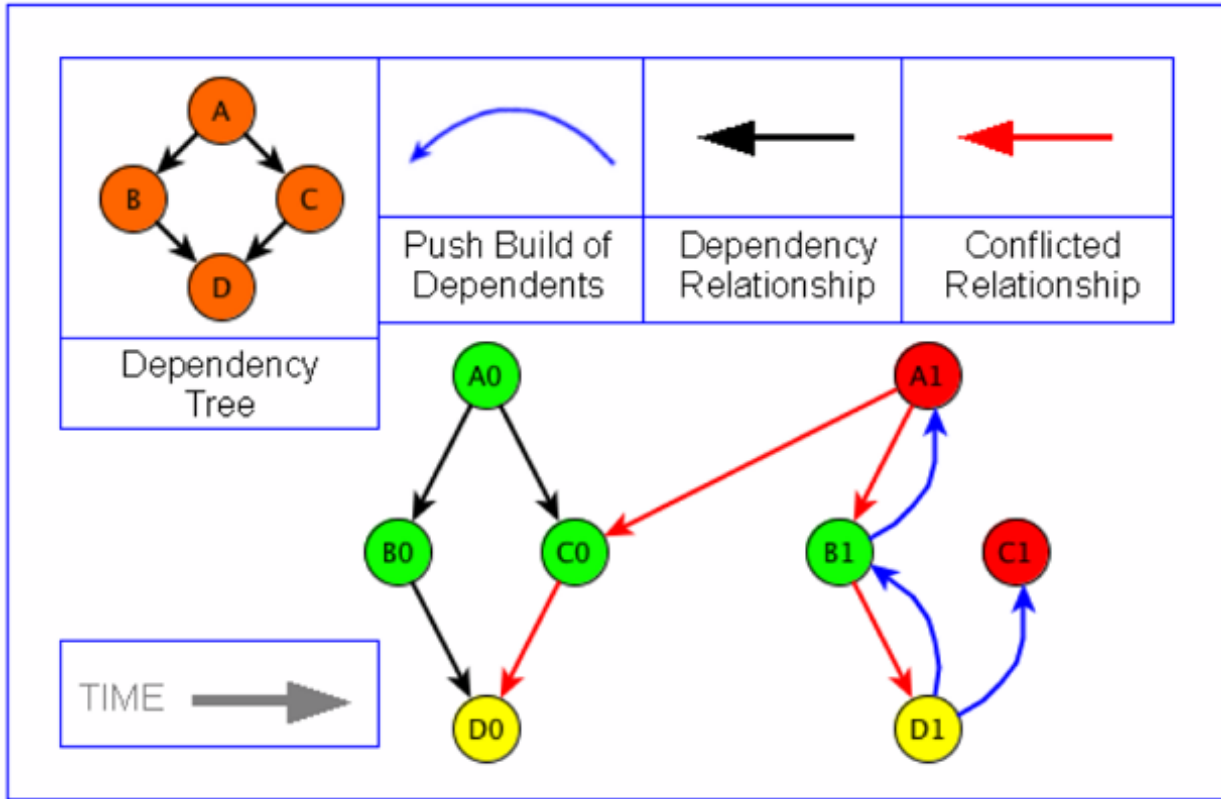
How the conflict strategy behaves is different for a pulled builds and pushed builds; however, the strategy is configured the same for both dependency trigger types.

Conflict Strategy with Push Dependencies

When project D needs to build (D1), it creates a request for builds B1 and C1. The builds of projects B and C result in a request for a build of project A (project A depends on a successful build of projects B and C). However, when project D builds, it pushes an new build of project C (C1). When the build of project C fails, the latest successful build of project C reverts back to build C0 -- this creates a mismatch with project D build versions, causing the

project A build to fail.

The conflict strategy is used to determine which version of project C to use when building project D (build D1) or to fail the build of project D.



See also Pushing Builds.

Chapter 31. Using Codestation Projects

A Codestation project models products (such as third-party tool kits and software libraries) not built by your organization. For example, if your web application requires Java 5, you configure a Codestation project and upload Java 5 into an artifact set. Once this is done, your other AnthillPro projects that require Java 5 will depend on the artifact set of this Codestation project. However, this Codestation project can never depend on another project.

Codestation projects also allow you to version the third-party tools or libraries within the artifact set. So if you decide to upgrade to Java 6, you can create a new Build Life (either on the project's Administration or Dashboard pages) and add the new artifacts. When the dependency is called during a build, AnthillPro will use your settings to determine which version to use (see Configure Dependencies). For example, it is possible for one component of your application to depend on Version 1 and a second component of your application to depend on Version 2 of the same Codestation project. This gives you fine-grained control of which versions of a third-party tool or library each AnthillPro project depends on.

Configure Codestation Project

1. Go to **Administration > New Codestation** icon.
2. Enter a **name**, give a **description**, select a **Life-Cycle Model**, and click **Save**.
3. If using **Maven**, you must assign a property by selecting the **Property** tab, clicking **New Property**, giving the property a **name** and **value**, and clicking **Save**.
4. Set **permissions** for who can use and edit the **Codestation** project. Select the appropriate **roles** on the **Security** tab. When done, click **Save**. See Setting Up Security.
5. **Create Build Life**. Because AnthillPro will not be building the project, you need to manually create a Build Life and upload new artifacts so they can be used by other projects. A typical approach would be to create a Build Life for every release of the third-party artifact managed, and then stamp that Build Life with the version of the library under management:
 - Click the **New** button under the **Build Life** section.
 - Input the **stamp** (e.g., if you are going to upload Java, give a stamp of Java 1.5, etc.) and click **Done**.
 - Select either **Upload File** or **Upload Many Files** (for uploading a .zip Archive).
 - Select the correct **artifact set**. The artifact set was inherited from the lifecycle model you selected in a previous step. If an existing artifact set does not exist, you may need to go to **System > Life-Cycle Models** and create one for the appropriate model.
 - **Directory Prefix**. Optionally, give a **base directory** at the root of the **artifact set** and **upload** the file.

When the file is uploaded a hash code is generated for the uploaded file to ensure that it is not modified and there are no corruptions when it is delivered to builds.

- Click **Upload**.
6. Once configured, the project's artifacts are available as dependencies of AnthillPro projects. See Configure Dependencies.
 7. For Codestation projects, a special status group reflecting your own approval system for third-party libraries is recommended. That way you can mark a new release of a library as *under review* or *experimental* while established

releases might be marked as *approved*. This makes it easy for a development team to use the recent release of the artifact that has been approved.

With this configuration in place, the Codestation project can be used in dependency schemes in the same way as any internal project. And when you need to start testing a newer version of the artifacts, simply come back and create a new Build Life and mark it appropriately.

- **Properties tab.** Add properties to the Codestation project to be used when resolving dependencies. See Managing Properties.
- **Status History tab.** Allows the administrator to promote this Build Life to additional statuses. It also tracks when those statuses were assigned. This can be useful when you have configured custom statuses on the Life-Cycle Model. Click **Add Status**, select the status from the drop-down menu, and click **Add**.

If you do not assign a status, you must configure your dependencies to use this Build Life's *Stamp*.

To add more statuses, repeat the process.

- **Dependencies tab.** Review which projects are using this Codestation Build Life. Once you configure an Ant-hillPro project to depend on this Build Life, it will be listed here.

Chapter 32. Configure Dependencies Tutorial

This tutorial shows how to facilitate code reuse in a controlled manner, and is based on the Release Reuse Equivalency principle first published by Robert Martin in Granularity [ht-

For dependency configuration instruction, see the main Configure Dependencies section.

Using the ABC Dependency project (hosted on Urbancode's public CVS server), this tutorial is designed to illustrate how dependencies are configured and how to troubleshoot problems. Projects B and C are each standalone (children) projects that generate Java libraries. Project A, the parent, uses those libraries (i.e., it depends on the other two projects).

1. To start, the `b.jar` and `c.jar` libraries are **removed** from **project A**.
2. Both jar files are configured as an independent AnthillPro project (Project B and Project C respectively).
3. Now, projects B and C compile fine, but without their libraries, Project A will not compile (as illustrated in the image below).



4. Drilling down into the build output log shows that the libraries from B and C were not available to the build.

```
build:
[mkdir] Created dir: C:\anthill4\agent\var\jobs\projects
        \Project_A__SVN_\build\classes
[javac] Compiling 1 source file to C:\anthill4\agent\var\jobs\projects
        \Project_A__SVN_\build\classes
[javac] C:\anthill4\agent\var\jobs\projects\Project_A__SVN_\source
        \java\anthill
        \dependency\a\A.java:8: package org.apache.log4j
        does not exist
[javac] import org.apache.log4j.Logger;
[javac]         ^
[javac] C:\anthill4\agent\var\jobs\projects\Project_A__SVN_
        \source\java\anthill
        \dependency\a\A.java:10: package
        anthill.dependency.b does not exist
[javac] import anthill.dependency.b.*;
[javac] ^
[javac] C:\anthill4\agent\var\jobs\projects
        \Project_A__SVN_\source\java\anthill
        \dependency\a\A.java:11: package
        anthill.dependency.c does not exist
[javac] import anthill.dependency.c.*;
[javac] ^
[javac] C:\anthill4\agent\var\jobs\projects
        \Project_A__SVN_\source\java\anthill
        \dependency\a\A.java:17: cannot find symbol
[javac] symbol   : class Logger
[javac] location: class anthill.dependency.a.A
[javac]     static private Logger log = Logger.getLogger(A.class);
[javac]         ^
[javac] C:\anthill4\agent\var\jobs\projects
        \Project_A__SVN_\source\java\anthill
        \dependency\a\A.java:19: cannot find symbol
[javac] symbol   : class B
[javac] location: class anthill.dependency.a.A
[javac]     private B b = null;
[javac]         ^
[javac] C:\anthill4\agent\var\jobs\projects
        \Project_A__SVN_\source\java\anthill
        \dependency\a\A.java:20: cannot find symbol
[javac] symbol   : class C
```

Dependencies and Artifact Sets

Any project that another project depends on is going to generate one or more collections of files (artifacts) the dependent project uses. In AnthillPro, each artifact collection is named to create an artifact set. For this set of projects, Project B and Project C are each going to have an artifact set for the libraries they produce, and are configured in a similar manner.

- **Artifact sets.** are also used to specify files that may be used in secondary workflows.
- **Example Artifacts:**
 - A **code library** used by multiple projects.
 - A platform-specific **client**.
 - **Documentation** of the shared library to be rolled into developer documentation (javadoc).
 - The **deployable executable** of a top-level project.

Because these are existing projects, the artifact sets have already been configured as part of a Life-Cycle Model. To view the artifact sets, follow the Artifact Sets link on the Life-Cycle Model page. The *lib* artifact set to hold the libraries and a *test* artifact set to manage test suites should be visible. See Using Life-Cycle Models.

Name: Example Life-Cycle Model
Description:

[Statuses](#) [Stamp Styles](#) [Artifact Sets](#) [Cleanup](#)

Artifact Sets are groupings for the artifacts (files) that are produced by a build. Artifact Sets allow the consumption of build artifacts in a more fine grained manner. This is important for deployments as well as for dependency management.

When used to support deployments, there is typically an artifact set for each deployment tier. So a typical three-tiered project with DB, APP, and WEB tiers would have corresponding DB, APP, and WEB artifact sets.

When used to support component based development and dependencies, there is typically an artifact set for each type of dependency artifact. Typically the run-time dependency artifact (jar, dll, etc.) files would be contained in an artifact set called "lib". A debug version of the run-time artifacts may be contained in an artifact set called "debug". The source code of the dependency may be contained in an artifact set called "source". And so on.

Create New

Name	Description	Operations
lib	holds the lib artifacts	
qtp-test	Mercury QTP Tests	
test	holds artifacts for unit tests	

Once the artifact sets are in place, go back to Project B (and C) and select this new group for those projects. You can also use this new artifact set for any other library-style project that might produce some tests. Now, you need to tell AnthillPro which libraries should be treated as part of the *lib* artifact set and when to publish the artifacts. Do this by editing the build job and workflow.

Edit the Dependency Build Job

In order for Project A to build successfully, the build jobs on Projects B and C must be edited so that the appropriate artifact sets are delivered. The process is the same for both projects.

1. From the Project B **Administration** page, select the **Edit Job** icon.

2. Select the **Edit** icon for the **Artifacts Deliver** step.
3. Select the **artifact set** to be delivered (**lib**). Click **Save**.

Artifact Deliver

Configure the Artifact Publisher in this section.

*denotes required field

Name: The name this step

Artifact Set: The artifact set you wish to publish (these are configured in your originating workflow)

Show Additional Options

- lib
- test
- web-app

Edit the Dependency Workflow

In order for Project A to build successfully, the workflows on Projects B and C must be edited so that the appropriate artifact sets are delivered. The process is the same for both projects.

1. From the Project B **configuration** page, select the **Edit Workflow** icon.
2. Select the **Artifacts** tab and specify what artifacts generated by this workflow are available for AnthillPro to retrieve and provide to other workflows and projects (the **lib** artifact set).

Main Definition Properties Dependencies **Artifacts** Triggers Security

Specify what artifacts generated by this workflow are available for Anthill to retrieve and later provide to other workflows and projects. These artifacts are retrieved by Anthill in the artifact publisher step.

Artifact Configurations

Artifact Set	Base Directory	Operations
lib	dist	

3. Identify the artifacts. Enter the **base directory** (relative to the directory that the checkout happened from) and one or more **artifact patterns** to include. Click **Save**.

- For a simple jar file, the base directory and the name of that file are sufficient.
- **Include Artifacts.** Lists artifacts to be retrieved. The wild card ****** indicates every directory and the wild card ***** indicates every file. So the pattern `dist/**/*` would retrieve the entire file tree underneath this dist directory. *Project and workflow properties are the only scripting allowed.* See Scripting.

To include symbolic links and empty directories, add the link/directory as part of your include pattern. See Configure Server Miscellaneous Settings.

- **Exclude Artifacts.** Patterns of artifacts that are in the include pattern but should be skipped. *Project and workflow properties are the only scripting allowed.* See Scripting.

Specify what artifacts generated by this workflow are available for Anthill to retrieve and later provide to other workflows and projects. These artifacts are retrieved by Anthill in the artifact publisher step.

New Artifact Config

Artifact Set	Base Directory	Operations
lib	dist	

Artifact Set: This is a name that is used to identify this set of artifacts. This same name will be used in steps that publish and resolve artifacts.

Base Directory: The location to base your artifact patterns on. This will be relative to the current working directory at the time the publish artifact step is run

Include Artifacts: List artifacts to retrieve. The wildcard ** indicates every directory and the wildcard * indicates every file. So the pattern dist/**/* would retrieve the entire file tree underneath this dist directory.

Exclude Artifacts: Patterns of artifacts that are in the include pattern but should be skipped.

Powered by anthillpro (Version 3.4.6.b4)

4. Once the artifacts are configured, future workflow executions of the build job will publish that library to AnthillPro's artifact store (Codestation). See Using Codestation Projects.

Configure the Dependent Project

Now that Projects B and C are delivering their artifacts to AnthillPro, instruct Project A to retrieve those artifacts.

1. Go to the Project A Main page (**Administration**) and select the **Edit Workflow** icon for the *project a build workflow*.
2. Select the **Dependencies** tab and choose Fail from the drop-down menu.
 - This will let you know a conflict has occurred which needs to be resolved before continuing.



3. Click the **New Dependency** button to create a dependency with Project B and Project C. Select the project and workflow creating the dependency and then determine how the dependency artifacts are to be provided.
4. Tell AnthillPro which **Artifact Set** to put in what location. The locations are relative to the workflow working directory. If an artifact set is not needed for this workflow (like the tests) simply do not check the box for it. In the example below, the artifacts for the *lib* artifact set (just the library produced by B) will be copied into the *lib* directory directly under the Project A root (shortly after checkout).

- **Artifact Retrieval.** 'Retrieve' indicates which artifact sets to retrieve from the dependency build life. 'Transitive' indicates whether the artifact sets with the same name from transitive dependency build lives should also be retrieved. 'Location' specifies the relative path(s) from the working directory to copy the dependency artifacts to. You can set multiple locations by using the add link. *Project and Workflow properties are the only scripting allowed; e.g., \${property:deps_dir}. (Use '.' for the current working directory, leave empty to remove).*

Project: Project B (SVN) - project b build workflow The dependency project

Build Life Criteria:

With Status:

With Stamp:

'Build Life Criteria' allows you to define which build life will be used to fulfill the dependency. The criteria for this is based on the status and/or stamp value. You can specify a stamp pattern using '?' for a single character and '*' for matching any sequence of characters.

Dependency Trigger:

None
 Pull Build
 Use Existing if Pull Fails
 Always Force
 Cascade Force
 Push Build

'Pull Build' pulls builds from the dependency relationship. Before this workflow runs, it will create a new build request for the dependency project. The build request may or may not create a new build life of the dependency based on if it had changes or not. If the request creates a new build life, that build life will be used to fulfill the dependency. If a build life is not created or the creation of the build life fails, the 'Build Life Criteria' will be used to find an existing build life. If the criteria is not specified then the build request for this workflow will fail.

'Push Build' pushes builds up the dependency relationship. Each time a build life of the dependency meets the criteria, a build request for this workflow is created.

Artifact Retrieval:

Artifact Set	Retrieve	Transitive	Location
lib	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	lib
test	<input type="checkbox"/>	<input type="checkbox"/>	

'Retrieve' indicates which artifact sets to retrieve from the dependency build life.

'Transitive' indicates whether the artifact sets with the same name from transitive dependency build lives should also be retrieved.

'Location' specifies the relative path(s) from the working directory to copy the dependency artifacts to. You can set multiple locations by using the add link. (Use '.' for the current working directory, leave empty to remove)

Save **Cancel**

5. Once both dependencies have been listed, **rebuild** both Project B and C to generate and deliver their artifacts for the first time. When a build of Project A is requested, the artifacts from Projects B and C that were delivered to the *lib* directory will be used to build Project A.

Part VII. Advanced Authoring

Many of the default settings for projects, workflows, jobs, and other AnthillPro resources can be customized and extended to meet your specific needs. A majority of the advanced authoring features are designed for users responsible for configuring AnthillPro for build and deployments. The tools allow these advanced users the ability to create reusable resources; copy configurations; use properties throughout the application; and control what information AnthillPro provides:

- **Projects.** Create, edit, copy, move, deactivate, activate, delete, import, and export projects on the Main Administration page.
 - **Workflows.** Create, edit, copy, deactivate, activate, delete, any workflow on the Main Administration page.
 - **Jobs.** Create, edit, copy, and delete jobs on the Main Administration page. In addition, you any job can be iterated.
 - **Reusing Workflow Definitions and Jobs.** Use library workflow definitions and library jobs are typically used to set up standardized processes across multiple projects.
 - **Using Life-Cycle Models.** Determine how a build is identified, the different stages a build must go through on its way to the end user, how artifacts are handled, and which clean-up policies are enforced.
 - **Notifications.** Have AnthillPro send e-mails, instant messages, and other notifications to select team members, or anyone who is interested in information the server generates.
 - **Properties.** Use properties to manage variables passed into commands, agent filters, and custom stamping algorithm templates. Properties can also be used to set up manual gates in your automated processes.
 - **Build Life Links.** Pass the URL of resources outside of AnthillPro to the Build-Life Reports tab.
-

Chapter 33. Managing Projects

All types of AnthillPro projects are created, edited, copied, moved, deactivated, activated, deleted, imported, and exported on the Main Administration page by selecting the appropriate icon. Projects may be created within folders or just under the Administration root folder. Only users with read/write appropriate permissions are able to perform these tasks. If you do not have the required permissions, contact your AnthillPro administrator. See Manage Security.

Importing and Exporting a Project

Importing and exporting projects is done from the Administration page. AnthillPro exports the project configurations as an XML file. The exported file may be used to transfer project configurations between multiple instances of AnthillPro, for troubleshooting purposes, etc. A project may be imported either into the Administration folder or into any sub-folder on the Administration page. Once successful, the imported project will have the same configurations as the original.

See also Import and Export Library Jobs and Import and Export Workflow Definitions.

Changing a Project's Source

Changing a project's source allows you to remove all source configuration from workflows within the project and then set up the project using a different SCM type, etc. **This process cannot be reversed.** If you choose to change a project's source type, you will need to reconfigure all the source for your originating workflows. If your reconfigured source is of a different type from what was originally used, you must also replace all the SCM-specific steps within jobs (both library and project) used by the project.

- **This feature is not intended to be used on any production project, except at the direction of Urbancode support.**

To change a project's source type:

1. Go to **Administration** and select the project.
2. On the project's page, click **Edit**.
3. Click the **Reset All Source Configurations** link.
4. Confirm that you want to change the Source Type. Note that once you click **Done**, all the source configurations for the project will be removed. You will then have to reconfigure the source for every originating workflow used by this project.
5. Select the originating workflow(s) associated with this project. You will be prompted with the Repository configuration page. Proceed with source configuration as you would for a new workflow. See Source Configuration.
6. You will also need to change any SCM-specific steps in your existing jobs if the repository type has changed. When modifying job steps, make sure to use the same step for the new SCM type.

Operational Projects

Use Operational (Non-Life-Cycle Based) Projects for administrative, operational, and system maintenance. Opera-

tional Projects are not connected to a Life-Cycle Model, and the project's workflows will not create a Build Life when a job is run.

If you have a lot of Operational Projects, you may want to include them in the Cleanup Policy. This will allow you to determine how long operational workflows and jobs are kept. See Cleanup Schedule.

Operational Projects are configured like regular projects on the Administration page, are managed identically to other projects, and visible on the Dashboard. Once the project is created, workflows and jobs are added to the project just as with any project.

In general, Operational Projects are used to automate processes ancillary to traditional build and release cycles, and can be thought of as secondary workflows that are not associated with a Build Life.

Chapter 34. Authoring Workflows

All types of workflows are created, edited, copied, copied to another project, deactivated, activated, deleted, on the Main Administration page by selecting the appropriate icon. Only those with the appropriate permissions are able to perform these tasks. If you do not have the required permissions, contact your AnthillPro administrator. There are numerous advanced-authoring tools that enable you to add additional automation, enforce process rules, and map AnthillPro processes to your existing strategies:

- **Stamping.** In AnthillPro, a stamp is a configurable name for the build number, build identifier, or version number used to identify a build. This allows you to mimic your current strategies.
- **Lockable Resources.** Use a Lockable Resource to specify a resource that needs to be identified and/or reserved (e.g., an agent or environment) when executing a workflow. AnthillPro prevents access to the resource when the lock is acquired, and queues all other requests for that resource until the lock is released upon workflow completion.
- **Workflow Tasks.** Workflow tasks allow you to set up manual gates, etc., that must be performed. For example, a task could be created for the QA lead to approve a build being promoted to another environment.
- **Triggers and Scheduled Builds.** With Triggers, builds may be performed in a number of ways. For Continuous Integration builds, it is ideal to run the build every time a file is checked in with a Repository Trigger. Other regular builds, such as nightly builds, are managed by schedules and kicked off by Scheduled Triggers.

Stamping

In AnthillPro, a stamp is a generic name for the build number, build identifier, or version number used to identify a build. Each stamp type may use a different algorithm to generate the stamp. Though stamps can be converted into labels (or baselines) on the corresponding source control, it is not required. In general, the stamp value should contain an incremental property with the syntax `${+property:<propname>}` on the project's originating workflow, making the property transparent to other originating workflows.

Though less robust, it is possible to generate a simple stamp using a short BeanShell script that returns the build ID. An example of this can be seen in the Setting Up a Build Process section.

Most projects will use numerous stamps: For example, a stamp may be applied to development builds using a source-control version number or may simply count build numbers. Another stamp may be applied only to more tested builds and use a releasable version number, etc. Because multiple stamps may identify the same build (within the Build Life), stamps are organized into stamp style for differentiation. See Applying Multiple Stamps to a Build.

Stamp styles are tied to stamp-value and/or stamp-context-script attributes that track version numbers and increment stamps through workflows. For example, a DEV stamp style would be used for marking development builds, a QA stamp style would be used for releasing to QA, etc. Typically, a new stamp style is used to identify when a build has reached a different stage of the application life-cycle. See Stamp Style.

- To use a stamp style, it must first be created on the project's Life-Cycle Model. See Using Life-Cycle Models.
- To use more than one stamp to track a build throughout its life-cycle, see Applying Multiple Stamps to a Build.

Once configured, all the stamps applied to a build can be viewed/traced on the Dashboard and Current Activity page.

Stamp Configuration Basics

Stamp configuration is (primarily) done with a string for each stamp style that may contain:

- **Foo**. Constant material.
- `${property:<propname>}`. Property material.
- `${+property:<propname>}`. Incrementing property.
- `${stampContext:<valuename>}`. Values generated from the stamp context script.

The `${+property:}` token can only reference originating-workflow properties and project properties, and is primarily used for a build-counter type functionality. It should not be referenced elsewhere in the workflow via `${property:}` notation because there is no guarantee that another concurrently running build has not incremented the value since the stamp was created. To have a separate counter for each originating workflow, it is usually necessary to create a workflow property.

See the Setting Up a Build Process section if you just want to return the build ID. In this way, there is no need to set any properties and the stamp will be assigned the Build Life number.

Stamp Context Scripts (Advanced)

Use stamp context scripts to generate values for variables in the stamp context when creating a stamp (for builds, etc.). The stamp context script is used by the stamping strategy to create a stamp within a workflow. The script should not modify any properties or objects, etc. Stamp context scripts are similar to normal AntHillPro scripts, except that the additional variable "stampContext" is provided an empty HashMap.

Any stamp configuration may be concatenated similar to: `${property:ProductName}-b${+property:buildCounter}.${stampContext:revision}` with a stamp context script. The `${stampContext:}` tokens are resolved by looking up values from the map generated by the stamp context script. For example, a stamp context script can be used to:

- Get the SCM revision number of the source that went into the build.
- Place the revision number in the map variable stamp context within the script.

It would look something like: `stampContext.put("revision", revisionNumber)`. The script can then be incorporate into the stamp by using the token `${stampContext:revision}`. The advantages over a simple `${bsh:}` in the stamp value is that a stamp context script can be reused by multiple workflows/stamps and can be written in scripting languages other than BeanShell. See Scripting.

Example Stamp Context Script: ChangeSet ID

The ChangeSet ID script below gets the most recent change set and is accessible to the stamp via `${stampContext:<name>}` syntax.

```
import com.urbancode.vcsdriver3.*;
import com.urbancode.anthill3.runtime.scripting.helpers.*;
int getMaxChangeSet(ChangeLog[] changelogArray) {
    int result = 0;
    for (int i = 0; i < changelogArray.length; i++) {
        ChangeSet[] changesetArray = changelogArray[i].getChangeSetArray();
```

```

        for (int j = 0; j < changesetArray.length; j++) {
            ChangeSet changeset = changesetArray[j];
            id = changeset.getId();
            // edit out the "r" character for svn
            if (id.startsWith("r")) {
                id = id.substring(1)
            }
            int num = (new Integer(id.trim())).intValue();
            if (num > result) {
                result = num;
            }
        }
    }
    return result;
}
int highestChangeset = getMaxChangeSet(ChangeLogHelper.getChangeLogArray());
// If there is no changelog, look up the most recent failed and good build
// and take the highest number. We assume that the version number will
// be the first stamp.
if (highestChangeset == 0) {
    lastSuccess = BuildLifeLookup.mostRecentSuccess();
    lastFailure = BuildLifeLookup.mostRecentFailure();
    if (lastSuccess != null) {
        int num = getMaxChangeSet(ChangeLogHelper.getChangeLogArray(lastSuccess));
        if (num > highestChangeset) {
            highestChangeset = num;
        }
    }
    if (lastFailure != null) {
        int num = getMaxChangeSet(ChangeLogHelper.getChangeLogArray(lastFailure));
        if (num > highestChangeset) {
            highestChangeset = num;
        }
    }
}
stampContext.put("changeset", ""+highestChangeset);

```

Create a Project Property for Stamping

1. Go to **Administration** and select the appropriate **project**.
2. On the **Project page**, select the **Properties** tab.
3. On the **Properties** tab, click the **New Property** button under the **Project Properties** menu.
4. **Name** the property and give it a **description (optional)**. Give the **value** as myStamp or another name that easily identifies it as a stamp.
 - **Pass to Builders.** Check Yes to automatically pass this property to applicable builders. Otherwise, check No. Automatically passing the property to the builder is not recommended, as it can lead to inaccurate stamps under certain conditions. If the stamp is going to be incorporated into the product, then pass something like this BeanShell `script${bsh:StampLookup.getLatestStampValue() }`, or `${bsh:StampLookup.getLatestStampValue("dev") }`, etc.
 - **Set in Environment.** Check Yes to automatically set this property in the environment for applicable builders.

Automatically setting the property in the environment is not recommended, as it can lead to inaccurate stamps under certain conditions.

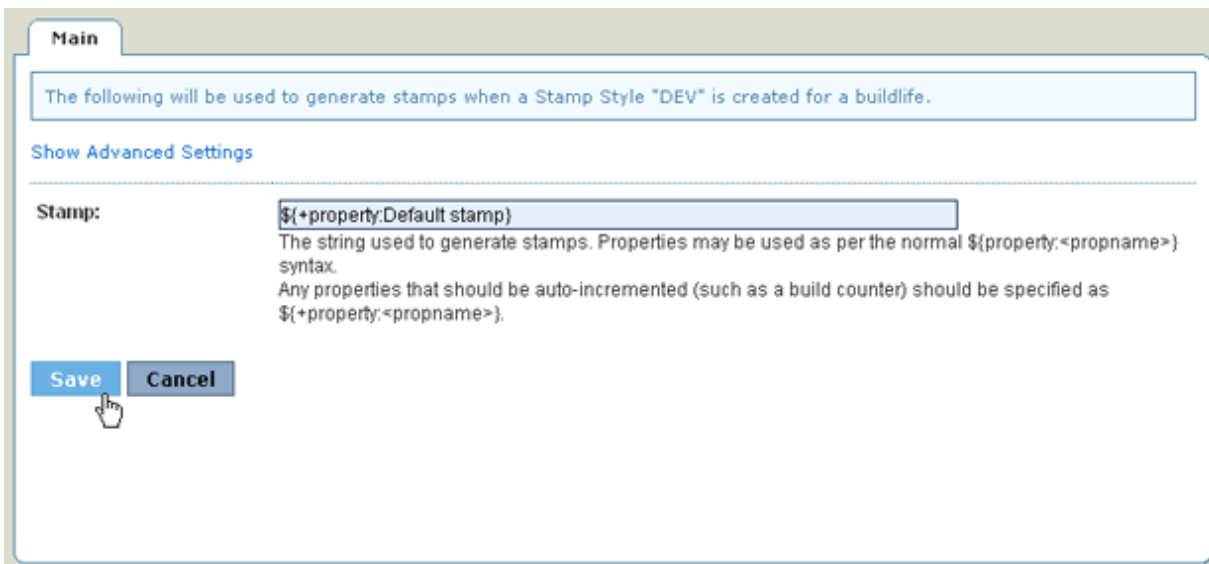
- **Value.** Input the property value.
- Click **Save**.

Add Stamp to Workflow

1. Go to **Administration** and select the **Originating Workflow** that you want to stamp.
2. Under the **Stamping Strategies** menu, click the **Create Value** icon under the **Action** menu to add a stamping strategy.

Style	Value	Action
DEV		+

3. On the **Stamping Strategy** configuration page, enter the **string** used to generate stamps.
 - Properties may be used as per the normal `${property:<propname>}` syntax.
 - Any properties that should be auto-incremented (such as a build counter) should be specified as `${+property:<propname>}`.



4. To set Advanced Settings, see Stamp Context Scripts (Advanced).
5. Click **Save**.

Advanced Settings (Stamp Context Script)

In order to use the advanced settings, first create a stamp context script and then add it to the appropriate workflow.

1. Go to **System > Stamp Context** under the **Script Library** menu.
2. On the **Stamp Context Scripts** page, click the **Create New** button.
3. **Name** the script and provide a **description (optional)**.
 - **Language.** Choose a scripting language from the drop-down menu. AnthillPro supports BeanShell, Groovy, and JavaScript.
 - **Script.** Input the script body. Set the properties in the stampContext map variable. These will then be accessible to the stamp via `${stampContext:<name>}` syntax. See Stamp Context Scripts (above) and Scripting.
 - Click **Save**.

Main

The following will be used to generate additional properties for Stamps.

*denotes required field

Name: *

Description:

Language: *

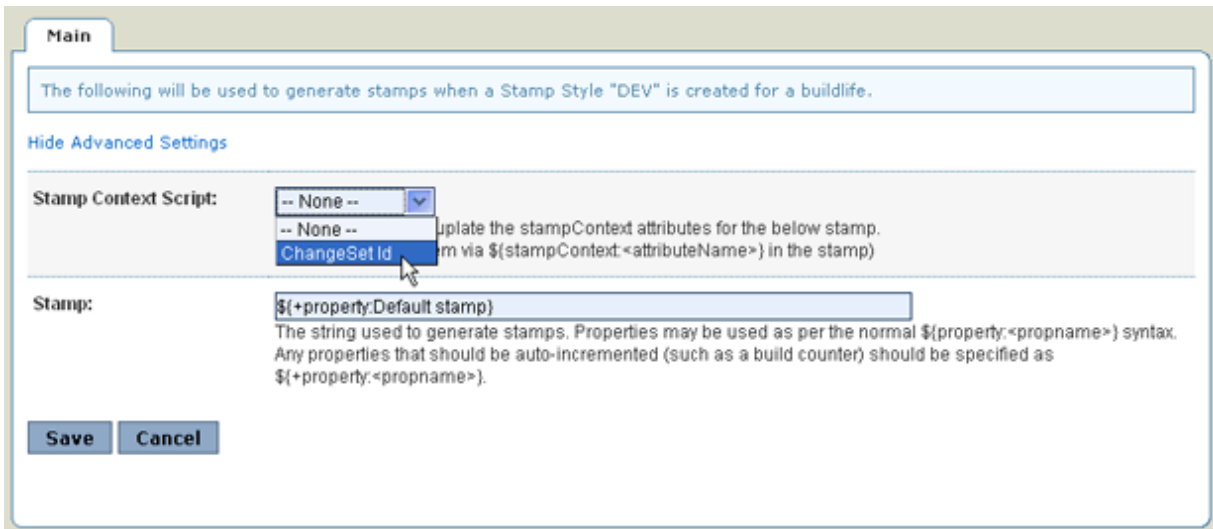
Script: * The script body. Set the properties in the "stampContext" map variable. These will then be accessible to the stamp via \${stampContext:<name>} syntax

```

1 import com.urbancode.vcsdriver3.*;
2 import com.urbancode.anthill13.runtime.scripting.helpers.*;
3
4 int getMaxChangeSet(ChangeLog[] changelogArray) {
5     int result = 0;
6     for (int i = 0; i < changelogArray.length; i++) {
7         ChangeSet[] changesetArray = changelogArray[i];
8         for (int j = 0; j < changesetArray.length; j++) {
9             ChangeSet changeset = changesetArray[j];
10            id = changeset.getId();
11            // edit out the "r" character for svn
12            if (id.startsWith("r")) {
13                id = id.substring(1);
14            }
15            int num = (new Integer(id.trim())).intValue();
16            if (num > result) {
17                result = num;
18            }
19        }
20    }

```

4. Go to the appropriate **Stamping Strategy** configuration page and select the **Show Advanced Settings** link.
5. From the drop-down menu, select the **script** used to populate the **stampContext** attributes for the stamp (you may access them via `${stampContext:<attributeName>}` in the stamp).
 - Enter the **string** used to generate **stamp**.
 - Click **Save**.



Applying Multiple Stamps to a Build

To identify the same build with different stamps, the **Create Stamp** step must be included in the secondary process (workflow) job. (While the step may be included anywhere within the job, most users include it after the **Populate Workspace** and **Get Changelog** steps if applicable.) For example, to stamp a build when it is sent to QA, the deploy (secondary) workflow must contain the Create Stamp step that tells AnthillPro to apply a new stamp whenever the workflow is performed. To apply additional stamps, each secondary workflow must contain the appropriate stamp style.

Adding New Stamp to Secondary Workflow

Add a new stamp to a build when a secondary process is run, typically to identify that the build has been sent to a different environment, etc. The process for applying a second (or third, etc.) stamp to a build is similar for stamping the build, except that the new stamp is tied to the secondary (non-originating) workflow used to send the build to another environment (QA, PROD, etc.). This is done by including a **Create Stamp** step in the deploy job.

1. **Create new Stamp Style.** Create the new stamp style that you want to apply to a build when the secondary workflow is run. See Using Life-Cycle Models.
2. **Create a Project Property.** In addition to the property for stamping the originating workflow, create a property for stamping the build when running a secondary workflow (process) with a different stamp. See Create a Project Property.
3. **Create the secondary workflow** that will apply the new stamp (via the job).
4. **Create Job to be run by the Secondary Workflow.** Once the secondary workflow has been created, configure the job that will deploy the artifacts.
 - Include the **Create Stamp** step near the beginning of the job.

Make sure to select the **Stamp Style** (from the drop-down menu) you want to apply when running this workflow.

5. Add job to workflow.
6. To add different stamps, for QAT, PROD, etc., follow Items 1-5 for each secondary process (workflow) that de-

loys the artifacts to the specific environment.

Creating Lockable Resources

Use a Lockable Resource to specify a resource that needs to be identified and/or reserved (e.g., an agent or environment) when executing a workflow. AnthillPro prevents access to the resource when the lock is acquired, and queues all other requests for that resource until the lock is released upon workflow completion.

After creating a Lockable Resource at **System > Lockable Resources** (see Configure Lockable Resources), it may be **edited**, **reset**, or **deleted** on the Lockable Resources page:

- **Edit.** Select the edit icon to change the Lock Resource configuration.
- **Reset.** Selecting the Reset icon removes all locks that have been acquired on the resource. For example, if the workflow is taking too long, or is hung up, clicking the Reset button will allow other workflows to acquire the resource. Resetting the Lockable Resource only effects workflows that have acquired the resource. Subsequent workflows will still use the Lockable Resource normally.
- **Delete.** A Lockable Resource configured on at least one workflow cannot be deleted.

Name	Description	Locks Used	Waiting	Used In Projects	Operations
PetShop Build		0 / 1	0	NET Pet Shop (Harvest) XPetStore (AccuRev)	
PetShop Deploy		0 / 1	0	NET Pet Shop (Harvest)	

When adding Locks to a workflow, there are two options:

- **Lock the Environment.** Select "Lock Environment" icon to have AnthillPro exclusively lock the environment when this workflow runs. See Exclusively Lock the Environment.
- **Resources to Lock.** Select the Add Resource link to have AnthillPro obtain a lock during workflow execution. Either choose a resource from the drop-down menu (see Use Lockable Resources); or select a resource dynamically with a BeanShell script.

If scripting a Resource Lock, it must return the name of a resource or a LockableResource object. When the workflow is run, AnthillPro will add the scripted Resource Lock to the Lockable Resources page. See Scripting.

Once a Lockable Resource is configured, it may be viewed (but not edited) by AnthillPro users with permissions to the Current Activity page.

Lockable Resources Prerequisites

- You must have administrative permissions. See Manage Security.
- At least one project must be active in AnthillPro.

Configure Lockable Resources

After creating a Lockable Resource, it must be added to a workflow so AnthillPro will lock a resource at workflow execution. See Use Lockable Resources.

1. Go to **System > Lockable Resources** under the Project Support menu.
2. On the **Lockable Resources** page, click **Create New**.
3. Configure:
 - **Name** the Lockable Resource.
 - **Description (optional)**. Give a description.
 - **Maximum Number Of Lock Holders**. Determine the maximum number of objects that can acquire this resource at any one time. Specifying "1" means that only one workflow can be run on the resource; specifying "5" means that a maximum of five different workflows can run concurrently using this resource.

It is also possible to override this setting for individual workflows. See Use Lockable Resources, Item Three.
4. Click **Save** then **Done**.
5. Proceed to Use Lockable Resources.

Use Lockable Resources

Once a Lockable Resource is configured (see Configure Lockable Resources), add it to a workflow on the Administration Workflow Main Page. This section covers the steps necessary to add an existing Lockable Resource to an existing workflow. To select a resource dynamically, see Scripted Lockable Resources.

1. Go to **Administration** and select the **workflow** the resource is to be acquired on.
2. On the **Workflow Main** page, select the **Add Resource** link.
3. Set the Lockable Resource:
 - **Explicit Resource**. Select an existing Lockable Resource from the drop-down menu. See Configure Lockable Resources.
 - **Scripted Resource**. If a Lockable Resource was selected above, skip this item. To dynamically set a Lockable Resource, see Scripted Lockable Resources.
 - **Exclusive**. Check "Yes" to have AnthillPro exclusively lock the resource. When enabled, AnthillPro will obtain an exclusive lock for this resource, and override the Maximum Number of Lock Holders set during resource configuration. See Configure Lockable Resources. When enabled, all other requests to acquire the resource will be queued.
4. Click **Save** then **Done**.
5. To add another Lockable Resource, repeat Items One through Four.

Exclusively Lock the Environment

Select the "Lock Environment" icon to have AnthillPro lock the environment when this workflow runs. When enabled, this option exclusively locks the environment, and ensures that no other workflow is able to run in the environment while this workflow is running. Additionally, if a request to run this workflow is made, and another workflow is currently running in the environment, this workflow will be placed on hold until the environment is free.

To turn exclusive environment locking off, select the icon again.

Creating Workflow Tasks

Assign the "task execute" permission to a user, on a workflow-by-workflow basis, to restrict that user's ability to execute another workflow. When a user is assigned this permission, they are allowed to only execute a workflow (such as a deployment, etc.) via a task (see Create and Use Workflow Choice Task and Create and Use Yes/No Task). Users with the "task execute" permission will not be allowed to manually run a secondary process, etc., and have no system permissions. This is useful in situations where AnthillPro users are only tasked with deploying, but not building, a project or workflow. To set a "task execute" permission, see Setting Workflow Permissions.

The task is created for originating workflows of Life-Cycle-Based projects, and typically result in a secondary workflow (such as a deployment) being run. See Create and Use Workflow Choice Task and Create and Use Yes/No Task.

Create and Use Workflow Choice Task

Use the Workflow Choice Task to require specific users to manually answer a question. When the question is answered with a workflow selected from all available workflows, that workflow will be run on the Build Life. To create a Task, add it to the job. When the job is run, the Task will appear on the Dashboard Tasks Tab for the users with the appropriate role (see Use Workflow Choice Task).

The Workflow Choice Task can be used with both an Embedded Job and a Library Job. (The Yes/No Task also may be used with a Library Job. See Create and Use Yes/No Task.) Because it requires a Build Life, the Task cannot be used in conjunction with Operational projects.

See also Execute Workflow via Task and Setting Workflow Permissions.

Workflow Choice Task Prerequisites

- You must have administrative permissions. See Manage Security.
- At least one Life-Cycle Based project must be active in AnthillPro. The Task will not work with Operational (Non-Life-Cycle Based) projects.
- The workflow to run the Task **and** the workflow(s) to be run when the Task is complete must be created.

Create Workflow Choice Task

While there is no restriction where the Workflow Choice Task is placed within the job, most users include it near the end of the job. For a typical build job, it is included after the Set Working Directory, Populate Workspace, Get Dependency Artifacts, and Build steps. The Workflow Choice Task may be added to either an originating or secondary (non-originating) workflow; however, it may only kick off a secondary (non-originating) workflow.

1. Go to **Administration** and select the **job** to add the Workflow Choice Task to. For detailed instructions on job creation.
2. Select the **Insert After** icon of the job step prior to where the Create Workflow Task step is to be included. For example, if inserting the Task step after the Build step, select the Insert After icon of the Build step.

Name		Type	Actions
Populate Workspace	CVS Populate Workspace		[edit] [up] [down] [left] [right] [stop] [cancel]
Get Changelog	CVS Get Changelog		[edit] [up] [down] [left] [right] [stop] [cancel]
Stamp	Create Stamp		[edit] [up] [down] [left] [right] [stop] [cancel]
Get Dependency Artifacts	Resolve Dependency Artifacts		[edit] [up] [down] [left] [right] [stop] [cancel]
build	Ant		[edit] [up] [down] [left] [right] [stop] [cancel]
JavaDocs	Report Publisher		[edit] [up] [down] [left] [right] [stop] [cancel]
ChangeLog	Changelog Publisher		[edit] [up] [down] [left] [right] [stop] [cancel]

3. Expand the **Miscellaneous** folder, select **Manual Workflow Choice Task**, and click **Select**.

4. Configure step:

- **Name** the step.
- **Question.** Give the question that a user must answer. The field may be scripted (advanced). See Scripting.
- **Role.** Select a role from the drop-down menu. Only one role may be selected. Any user assigned the selected role will be able to answer the question by selecting one of the available workflows.

To restrict who can answer the question and execute the workflow, See Execute Workflow via Task.

- **Workflow Choices.** Check the secondary workflow(s) that will be displayed for the user to choose from (see Use Workflow Choice Task). For example, if multiple workflows are available to the user, one can be used to send the artifacts to QA, or one used to send the artifacts to QAT, etc.

Only secondary workflows associated with the project are available: i.e., a Library Workflow cannot be kicked off with this Task. For details on creating secondary workflows.

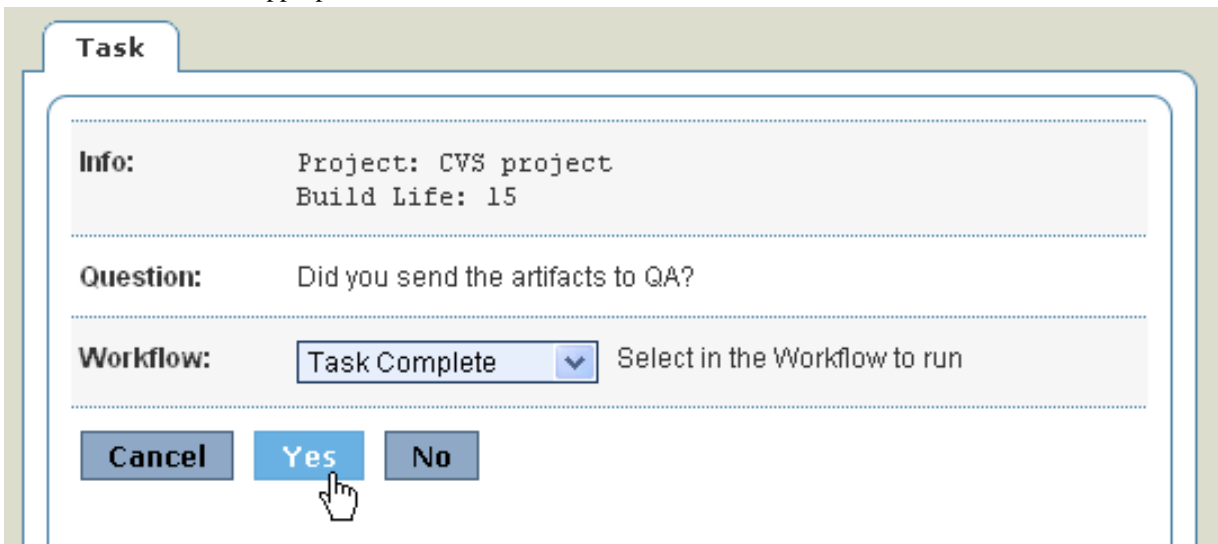
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** Select the pre-condition script which must pass before the step will execute. *Editing an existing script will effect all projects that use the script.*
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

5. Click **Save**.

6. For usage, see Use Workflow Choice Task.

Use Workflow Choice Task

1. Run the workflow that contains the job configured in Create Workflow Choice Task.
2. Once the workflow is complete, go to **Dashboard > Tasks**. If you have the appropriate role, the Task will be listed.
3. Under the **Description** menu, select the appropriate Task.
4. In the Task dialog box, answer the question configured in Create Workflow Choice Task.
 - **Workflow.** Select the workflow to be run from the drop-down menu. Because there are multiple options, make sure you know which workflow to choose before completing the Task.
 - Click **Yes** to add the appropriate workflow.



5. Click **Yes** again to kick off the workflow. If you click **No** the workflow will not run. Clicking **Cancel** will return you to the main Tasks tab.
6. To view the status and history of every Task, see Task History Tab.

Create and Use Yes/No Task

Use a Yes/No Task to require a specified user role to perform a manual procedure in order to kick off another workflow. To create a Task, add it to the job. When the job is run, the Task will appear on the Dashboard Tasks Tab for the users with the appropriate role (see Use Yes/No Task).

For example, use a Task to verify that a QA lead reviewed the automated test logs. When "Yes" is selected (indicating approval) a workflow that deploys the application to QA is triggered.

The Yes/No Task may be used with both an Embedded Job (see Create Yes/No Task with Embedded Job) and a Library Job (see Create Yes/No Task with Library Job). Because it requires a Build Life, the Task cannot be used in conjunction with Operational projects.

See also Execute Workflow via Task and Setting Workflow Permissions.

Yes/No Task Prerequisites

- You must have administrative permissions. See Manage Security.

- At least one Life-Cycle Based project must be active in AnthillPro. The Task will not work with Operational (Non-Life-Cycle Based) Projects.
- The workflow to run the Task **and** the workflow to be run by the Task must already be created.

Create Yes/No Task with Embedded Job

For a typical build job, the Task is included after the Set Working Directory, Populate Workspace, Get Dependency Artifacts, and Build steps. Most users include the step near the end of the Job. The Yes/No Task may be added to either an originating or secondary workflow; however, it may only kick off a secondary (non-originating) workflow for the same Build Life.

1. Go to **Administration** and select the **job** to add the Yes/No Task to. For detailed instructions on job creation.
2. Select the **Insert After** icon of the job step prior to the point where the Yes/No Task step is to be included. For example, if inserting the Task step after the Build step, select the Insert After icon of the Build step to add the Task step after the Build step.

Name	Type	Actions
Populate Workspace	CVS Populate Workspace	[Edit] [Move Up] [Move Down] [Refresh] [Stop] [Delete]
Get Changelog	CVS Get Changelog	[Edit] [Move Up] [Move Down] [Refresh] [Stop] [Delete]
Stamp	Create Stamp	[Edit] [Move Up] [Move Down] [Refresh] [Stop] [Delete]
Get Dependency Artifacts	Resolve Dependency Artifacts	[Edit] [Move Up] [Move Down] [Refresh] [Stop] [Delete]
build	Ant	[Edit] [Move Up] [Move Down] [Refresh] [Stop] [Delete]
JavaDocs	Report Publisher	[Edit] [Move Up] [Move Down] [Refresh] [Stop] [Delete]
ChangeLog	Changelog Publisher	[Edit] [Move Up] [Move Down] [Refresh] [Stop] [Delete]

3. Expand the **Miscellaneous** folder, select **Manual Yes/No Question Task**, and click **Select**.
4. Configure step:
 - **Name** the step.
 - **Question.** Give the question that a user must answer. The field may be scripted. See Scripting.
 - **Role.** Select a role from the drop-down menu. Only one role may be selected. Any user assigned the selected role will be able to answer the question.

To restrict who can answer the question and execute the workflow, See Execute Workflow via Task.

- **Workflow to Run.** Select the workflow that will run when the question is answered. Only when the user answers "Yes" to the question will the secondary workflow execute on the Build Life.
- **Environment.** Select the environment the secondary workflow will run in (once it has been kicked off by the manual Task). If "Any Environment" is chosen, AnthillPro will run the workflow on the first available agent associated with the project. If the workflow to be run requires a resource only available in a single environment, select that environment. See Environments.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.

- **Pre-Condition Script.** Select the pre-condition script which must pass before the step will execute. *Editing an existing script will effect all projects that use the script.*
- **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
- **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
- **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

5. Click **Save**.

6. For usage, see Use Yes/No Task.

Create Yes/No Task with Library Job

Creating a Yes/No Task for a Library Job follows much the same process described in Create Yes/No Task with Embedded Job; however, because the Library Job may be used by multiple projects, the workflow to be run and the specific environment must be determined by a short script.

To use the Task with a Library Job, you should already be familiar with AnthillPro scripting. See Scripting.

1. Follow the directions for creating a Library Job.
2. Go to **Administration** and select the **Library Job** to add the Yes/No Task to.
3. Select the **Insert After** icon of the job step prior to the point where the Yes/No Task step is to be included. For example, if inserting the Task step after the Build step, select the Insert After icon of the Build step to add the Task step after the Build step.
4. Expand the **Miscellaneous** folder, select **Manual Yes/No Question Task**, and click **Select**.
5. Configure step:
 - **Name** the step.
 - **Question.** Give the question that a user must answer. The filed may be scripted (see Scripting).
 - **Role.** Select a role from the drop-down menu. Only one role may be selected. Any user assigned the selected role will be able to answer the question.

To restrict who can answer the question and execute the workflow, See Execute Workflow via Task.

- **Workflow Script.** Give the script (see Scripting) that will determine the workflow to run when the question is answered "Yes". The script should return the name of the workflow or the workflow object to run.

For example, enter

- **return "Deploy";** to kick off the secondary workflow named Deploy when the question is answered "Yes";

or

- `return ProjectLookup.getCurrent().getWorkflow("Deploy");` to first look up the current project and kick off the secondary workflow named Deploy when the question is answered "Yes".
- **Environment Script.** Give the script (see Scripting) that will determine the environment to run the workflow when the question is answered "Yes". The script should return the name of the environment.

For example, enter

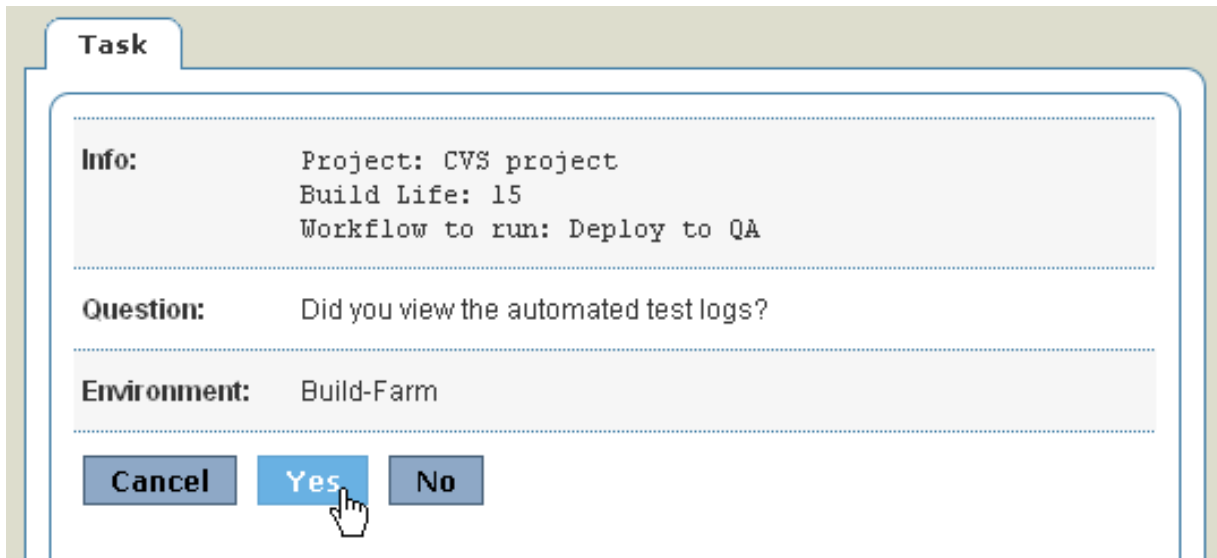
- `return "QA";` to run the secondary workflow in the QA environment when the question is answered "Yes". See Environments.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** Select the pre-condition script which must pass before the step will execute. *Editing an existing script will effect all projects that use the script.*
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

6. Click **Save**.

7. For usage, see Use Yes/No Task.

Use Yes/No Task

1. Run the workflow that includes the job configured in either Create Yes/No Task with Embedded Job or Create Yes/No Task with Library Job. See Run Builds for originating workflows or Run Deployment for secondary workflows.
2. Once the build is complete, go to **Dashboard > Tasks**. If you have the appropriate role, the Task will be listed.
3. Under the **Description** menu, select the appropriate Task.
4. In the Task dialog box, answer the question configured in Create Yes/No Task with Embedded Job or Create Yes/No Task with Library Job. Click **Yes** to kick of another workflow. If you click **No** the workflow will not run. Clicking **Cancel** will return you to the main Tasks tab. Once a Task has been completed, it will be removed from the Tasks tab.



Task

Info: Project: CVS project
Build Life: 15
Workflow to run: Deploy to QA

Question: Did you view the automated test logs?

Environment: Build-Farm

Cancel Yes No

5. To view the status and history of every Task, see Task History Tab.

Workflow Triggers and Scheduled Builds

With Triggers, builds may be performed in a number of ways. For Continuous Integration builds, it is ideal to run the build every time a file is checked in with a Repository Trigger. Other regular builds, such as nightly builds, are managed by schedules and kicked off by Scheduled Triggers.

Builds (and deployments) may also be started based on arbitrary events in the system using a scripted Event Trigger. For example, event triggers can be used to look for actions such as a completed build of a related project or completion of an originating workflow with the auto-deploy flag set to "true".

If you use Subversion, see also Using Triggers with Subversion.

AnthillPro supports three trigger types:

- **Event Triggers.** Looks for actions automatically thrown by AnthillPro and initiates another action (e.g., initiates a deployment on successful completion of an originating workflow). See Event Triggers.
- **Repository Triggers.** Kicks off a build when changes are committed to source. Most commonly used for CI. See Repository Triggers.
- **Scheduled Trigger.** Each schedule runs on its own timer. The build is registered with the schedule and kicked off when the schedule fires. See Scheduled Triggers.

Triggers and Scheduled Builds Prerequisites

- You must have Administrative permissions. See Manage Security.
- Prior to adding an Event Trigger to the workflow, the Event Script must be created. See Event Scripts.
- If you are unfamiliar with AnthillPro scripting, see Scripting and **Tools > Developer Tools** before creating an Event Trigger.

- See also Use Agent Filters and Quiet Periods.

Event Triggers

Event Triggers use scripts to create an Event Filter that listens to events passing through the AnthillPro Event Service. When the Event Script detects an event (e.g., that a dependency of the trigger's workflow is used; a completed build of a related project; or a completed originating workflow with the auto-deploy flag set to "true"; etc.), it can then trigger another action by the AnthillPro server.

In general, once configured, the scripts are immediately active and listening for events. The trigger, workflow, and project parameters provide references to the Event Trigger itself, as well as the workflow and project it belongs to. The AnthillPro API exposes its event service so the Trigger may be configured to listen for specific events. See Event Scripts and Scripting.

See also Use Agent Filters and Quiet Periods.

Create Event Trigger

1. Go to **Administration** and select the **workflow** the trigger is to be added to.
2. Select the **Triggers** tab, and click the **New Trigger** button.
3. Select **Event Trigger** from the drop-down menu and click **Select**.
4. Configure trigger:
 - **Name** the trigger.
 - **Force**. Check the box to force a build every time this trigger executes.
 - **Event Script**. Select the script that returns an EventListener. Available parameters are the workflow to be triggered (workflow); this project (project); and this trigger (trigger). See Event Scripts.
5. Click **Save** then **Done**.

Repository Triggers

For Continuous Integration builds, it is ideal to build every time a file is checked in. In this case, a post-commit script is added to the SCM that alerts AnthillPro of source changes. Once the Repository Trigger is active for a workflow, every commit of source changes will initiate a build.

See also Use Agent Filters and Quiet Periods.

1. Go to **Administration** and select the **workflow** the trigger is to be added to.
 - If using Subversion, triggers can be set up at the repository level. Once done, they are available to all workflows using that repository. An individual workflow trigger does not need to be created. See Subversion Repository Trigger Configuration.
2. Select the **Triggers** tab, and click the **New Trigger** button.
3. Select **Repository Trigger** from the drop-down menu and click **Select**.

4. Configure trigger:

- **Name** the trigger.
- **Force**. Check the box to force a build every time this trigger executes.

5. Click **Save** then **Done**.

Scheduled Triggers

Add a Scheduled Trigger to workflows in a similar manner to the Repository Trigger. Each Schedule Trigger runs on its own timer that polls the SCM for changes. If changes are detected, the build is registered with the schedule and kicked off when the schedule fires.

Scheduled Triggers give the option to force a build regardless of whether source changes have occurred. Unless the forced option is set, AnthillPro will not initiate a build if no source changes are detected.

See also Use Agent Filters and Quiet Periods.

Create Schedule for Trigger

A schedule must be created prior to adding to the trigger. To do so, go to **System > Schedules** under the Project Support menu. When creating a schedule, make sure to give it a name and description that makes it easy for others to identify (e.g., Name: Hourly Schedule; Description: kicks off a build every hour).

To create a schedule, see Create Schedules.

Add Schedule to Workflow Trigger

Before continuing, make sure you have already created a schedule. See Create Schedule for Trigger.

1. Go to **Administration** and select the **workflow** the trigger is to be added to.
2. Select the **Triggers** tab, and click the **New Trigger** button.
3. Select **Scheduled Trigger** from the drop-down menu and click **Select**.

4. Configure trigger:

- **Name** the trigger.
- **Schedule.** Select a schedule from the drop-down menu. See Create Schedule for Trigger.
- **Force.** Check the box to force a build every time this trigger executes, even if source changes have not been detected.


5. Click **Save** then **Done**.

Chapter 35. Administrating Jobs

Jobs are created, edited, copied, copied to another project, and deleted on the Main Administration page by selecting the appropriate icon. Any job may also be iterated (to allow for multi-platform builds, etc.) to run multiple times. AnthillPro also provides a way for you to create templated jobs, called a Library Job. Once a Library Job has been configured, it can be used by multiple projects or can be turned into a project-specific job. When used in concert with Reusable Workflow Definitions, the Library Job can provide fine-grained control over your AnthillPro processes, as well as cut down on the amount of configuration necessary when editing existing and creating new projects. See Reusing Workflow Definitions and Jobs for more.

Reusing Job Steps

It is possible to copy steps between jobs in the same project, between jobs in different projects, or past them within the same job. This allows you to reuse configured steps. Once the step has been copied and used to the new location, you can edit the step's settings to customize it for the particular job.

1. Select the **Copy Step** icon (under the Actions menu) of every step to be copied.
2. Once all the steps have been added to the clipboard, navigate to the destination project and then paste the step(s) into the appropriate job by selecting either the **Paste** or **Paste and Remove** icon on the clipboard.
3. When pasted, the step(s) will appear at the end of the existing job. To move the step, use the grab tool  (to the right of the step name) and drag the step to the desired location. To clear the clipboard, click the **Remove** icon.
4. Make any necessary changes to the copied step.
5. To clear the clipboard, click the **Remove** icon.

Iterate a Job

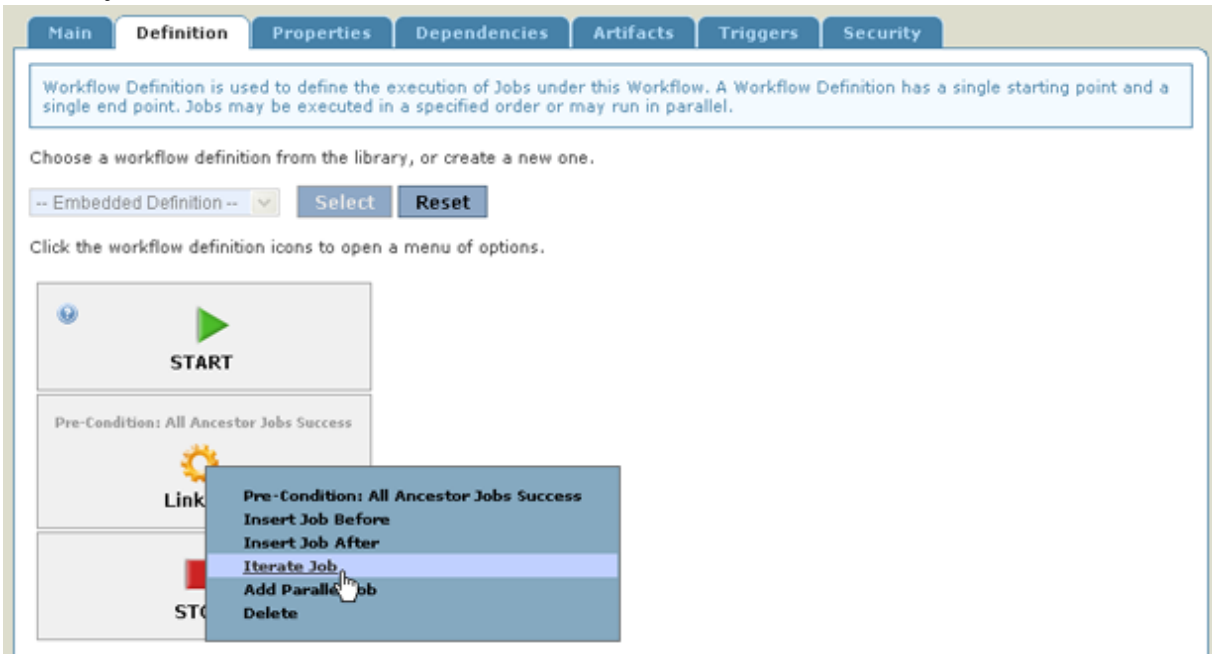
Use job iterations to run the same job multiple times. When used in conjunction with Job Iteration Properties, you can set the parameters for a single job to run with slight variations each time. For example, if you want to run a deployment on several machines, you can set a property on each iteration that identifies the machine (with an AnthillPro agent installed on it) to run the deployment on. Or, in the case of a multi-platform build, you can set iterations that will iterate the build across each platform. Before you begin, make sure:

- You must have AnthillPro administrative privileges to iterate a job. See Manage Security.
- A Project with an originating workflow and at least one job must already be active.
- If you are planning on using job-iteration properties to determine which agent each iteration will run on, you will need to create a new Agent Filter Script. See **Tools > anthill3-dev-kit.zip > Scripting > Example: Select Agents Based on Job-iteration Properties**.
- Corresponding properties must be set on the agent(s) so that when the agent filter script runs, it will select the correct agent(s). See Configure and Edit Agent Filters.

Configure Job Iterations:

1. Go to **Administration** and select the **workflow** that contains the job to be iterated.

2. On the workflow page, select the **Definition** tab.
3. Click the job to be iterated and select **Iterate Job** from the menu.



4. Configure Job Iterations:

- **Job.** This is the name of the job to be iterated. This field is automatically populated by AnthillPro.
- **Iterations.** Give the number of times this job is going to be iterated. Use **-1** to iterate a number of times *equal to the number of agents returned* by the job's agent filter. If -1 is used, Unique Agents must be set to **Yes**. For example, if the agent filter returns 4 agents, setting the iterations to -1 will iterate the job four times.
- **Unique Agents.** To have each iteration run on a unique (different) agent, check **Yes**. If the number of iterations set above is -1, you must check Yes here, otherwise the iterated job will fail.

If you are planning on using job-iteration properties to determine which agent each iteration will run on, make sure you check **No** -- otherwise your jobs may fail. See Job Iteration Properties for more.

- **Run Parallel.** Check Yes to run in parallel (default is No). Depending on your system, running parallel jobs may not be possible; however, running parallel jobs can significantly decrease build times.

5. Click **Set Iteration**.
6. To set job iteration properties, see Job Iteration Properties.

Chapter 36. Reusing Workflow Definitions and Jobs

Library workflow definitions and library jobs are typically used to set up standardized processes across multiple projects.

The **library job** can be used either by a project workflow or a library workflow definition. When added to a project during workflow configuration, the job is added on the workflow Definition page as an embedded job. This allows each individual workflow to use a combination of library jobs and project jobs, and provides flexibility while still allowing you to reuse many of the same processes.

The **library workflow**, configured similar to a project workflow, can only use library jobs as part of its definition. When added to a project during workflow configuration, the workflow definition is added on the workflow Definition page as a library definition.

Library jobs and workflow definitions do not necessarily have to be placed in the Job Library/Workflow Library folder -- they can be placed anywhere within the Admin tree. Additionally, any jobs or workflow definitions (on the Administration page) not in a folder or as part of a project subdirectory are library jobs and workflow definitions.

See Creating Library Jobs and Creating Workflow Definitions.

Creating Library Jobs

Once configured, the library jobs are added to workflows of any existing project on the workflow Definition tab. Library jobs may also be added to library workflows, and will be used by any project configured to use that Workflow Definition (see Creating Workflow Definitions). Additionally, Library Jobs may also be created by copying an existing project job.

Set Up a New Library Job

1. Go to **Administration > Job Library**.
2. On the **Job Library** page, click the **New Job** link.
3. Configure the job.

*If the job is to be used in an **Operational Project**, do not select a **Life-Cycle Model** or **SCM type**. See *Operational Projects*.*

4. Click **Save**.

Add Library Job to a Project Workflow

1. Go to **Administration** and select the **project** that will use this job.
2. Select the **workflow** that this job will be added to as part of the definition.
3. Select the **Definition** tab.
4. Select **Embedded Definition** from the drop-down menu, and click **Select**.

5. Click the **START** icon to insert the job.
6. Select the **job** from the drop-down menu, select the **Pre-condition Script**, and click **Insert Job**.

To use a library job as part of a library workflow definition, see [Creating Workflow Definitions](#).

Import and Export Library Jobs

To export a Library Job:

1. Go to **Administration**.
2. Click the **Export Job** icon of the job to be exported.
3. The Job configuration will be exported to the specified directory (or opened up in an XML editor).

To import a Library Job:

1. Go to **Administration** > **Import Project** icon of the Library Job folder.
2. Import the file containing the XML description, or paste the XML description into the text field.
3. Click **Import**.

See also [Import and Export Workflow Definitions](#).

Creating Workflow Definitions

Once configured, Workflow Definitions are added to existing projects as part of the project-workflow creation process (on the project-workflow Definition tab).

Set Up New Workflow Definition

1. Go to **Administration** > **Workflow Library**.
2. On the **Workflow Library** page, follow the **New Workflow** link.
3. Configure the **Workflow Definition** and click **Save**.
4. Add a **job** to the Workflow Definition. Only jobs created in the Job Library may be added to this workflow definition. See [Creating Library Jobs](#).

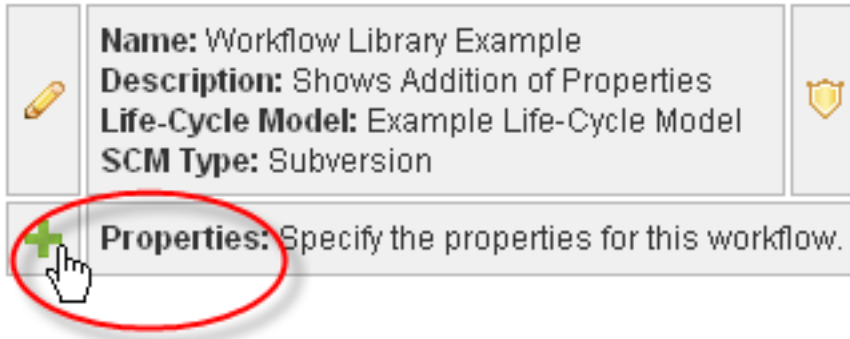
Specify Workflow Definition Properties

Specify any Properties to be passed to all workflows using this library workflow. You will need to define a property on the library workflow, the value of which will be different for each project workflow that uses the library definition.

For example, you have two projects that use a library workflow for builds. To have each project use the correct build script, you would define a property on the library workflow. Then, when you configure the projects that use this

workflow definition you would assign a value to the property.

1. Go to **Administration > Workflow Library** and select the appropriate Workflow Definition.
2. Select the **Add Property** icon.

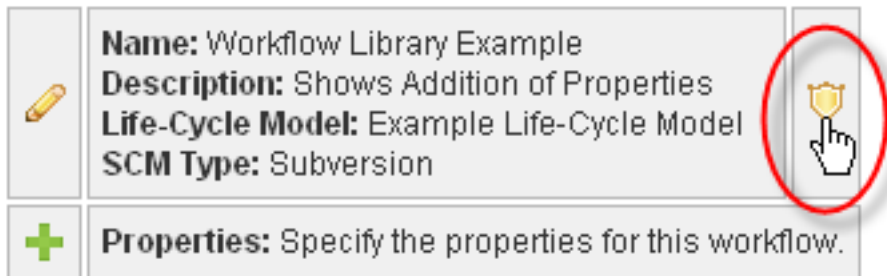


3. **Name** the Property and click **Save**.

Set Workflow Definition Security

Configure the permissions for the Workflow Definition. Determine what user roles can perform which actions on the Workflow Definition. See Manage Security.

1. Go to **Administration > Workflow Library** and select the appropriate Library Definition.
2. Select the **Security** icon.



3. Determine permissions. See Manage Security.
4. Click **Save** then **Done**.

Add Workflow Definition to a Project

1. Go to **Administration** and select the workflow that the **Workflow Definition** is to be added to.
2. Select the **Definition** tab.
3. Click the **Reset** button and select the **Workflow Definition** from the drop-down menu.

4. Click **Select**.
5. Select the **Properties** tab if Property has been specified on the Workflow Definition. See Specify Workflow Definition Properties.
6. Select the Property created in the Specify Workflow Definition Properties step.
7. Configure Property:
 - **Name.** The name of this Property has been inherited from the Workflow Definition. Do not change the name.
 - **Description.** Provide a description for this Property to be shown when prompting users for a value.
 - **Default Value.** Give the value for this property.
 - **User May Override.** Check the box if the user may specify a new value when running the workflow.
 - **Label.** Provide a label for this Property to be shown when prompting users for value (leave blank to use the Name as the label).
 - **Is Required.** Check the box to require a non-empty value for this property when running this workflow.
 - **Allowed Values.** Give the values users are allowed to select for this property (leave blank for no value restriction). Separate each value by entering it on its own line.
8. Click **Save**.

Import and Export Workflow Definitions

To export a Library Workflow:

1. Go to **Administration**.
2. Click the **Export Workflow** icon of the workflow to be exported.
3. The workflow configuration will be exported to the specified directory (or opened up in an XML editor).

To import a Library Workflow:

1. Go to **Administration** > **Import Project** icon of the Library Workflow folder.
2. Import the file containing the XML description, or paste the XML description into the text field.
3. Click **Import**.

See also Import and Export Library Jobs.

Chapter 37. Using Life-Cycle Models

A Life-Cycle Model allows you to create a reusable template that maps your organizational structure. For example, a typical set-up would be DEV, QA, PROD process (i.e., pipeline or life-cycle): a build starts out in development, is deployed to quality assurance for testing, and then finally sent to production for release. You would configure the Life-Cycle Model to apply a new status when a build is sent to QA, and one when the build is sent to PROD. Likewise, a stamp style (essentially a build identifier) can be applied to each build that corresponds to the status. This enables you to know exactly which build is in which environment because the status and stamp are recorded on the Dashboard.

Life-Cycle Models, then, give you control over how a build is identified, the different stages a build must go through on its way to the end user, how artifacts are handled, and which clean-up policies are enforced.

Life-Cycle Models are closely associated with AnthillPro's Build Life concept: Much of what you configure in the Life-Cycle Model determines how a Build Life is identified and used. See Build Life for more.

Once a Life-Cycle Model is created, it may be used for multiple projects with similar Life-Cycles without having to reconfigure a list of Statuses, Stamps, Artifact Sets, and Cleanup for the new project. This enables you to create system-wide standards that every team must use when configuring projects.

Creating a Life-Cycle Model

Make sure you have administrative permissions before continuing. See Manage Security.

1. Go to **System > Life-Cycle Models** under the Project Support menu, and click the **Create New** button.
2. Give the Model a **name** and a brief **description (optional)**. Click **Save**.

Status

Success and failure statuses are applied by default (because they are required), but common status names such *PROD* or *QA* can be created and associated with a color (using a drop-down menu) for ease in spotting Build Lives at a given status. As an indication of what stage a Build Life has reached, status names similar to those of environment groups can be useful in identifying which environment groups a Build Life has been deployed to.

If you have a number of statuses in your Life-Cycle Models, you can reorder them using the drag-and-drop tool so that the ones you are most interested in appear at the top of the list. The order set here will be displayed on the Dashboard Workflow page once a workflow is complete.

1. To create a new status, select the **Status** link from the **New Life-Cycle Model** main page.
2. Click the **Create New** button.
3. **Name**, give a **description** of, and assign a **color** to the new status. When selecting a color, you can use the up/down keys to quickly scroll through the picker.

New Status

A status is an indication of what stage in a build life has been reached. It is often used to represent promotion levels like, qa, user acceptance, or production. Status names similar to the short names of server groups can be useful in easily identifying which server groups a build life has been deployed to.

*denotes required field

Name: * The name of the status.

Description: The description of the status.

Color: The color of the status. Selecting a color will display it below.

4. Click **Save**.

5. Arrange order by using the Move Status Up and/or Move Status Down icons under the Operations menu.

Stamp Style

Stamps are essentially numbers or labels used to identify a build, and are used to help identify a particular build. Stamp styles (or stamp types), then, are used to apply different stamps to a single build, and allow you to help track a build throughout its life-cycle. While most projects will only need a single stamp representing the build number, many AnthillPro users find it helpful to apply multiple stamp styles to specific projects, such as those released into production.

Typically, apply a new stamp to a build when it is promoted (or deployed or released) to a new stage in the life-cycle by creating multiple stamp styles. So a DEV stamp style would be used to identify development builds; a QA stamp style would be used for identifying the build when it is sent to QA; and a PROD (production) stamp style would be used to identify when the build has been released into production.

In this way, you can set up AnthillPro to stamp a build according to the protocol of each individual environment (or stage) that the build goes through. For example, a single build may have 3 (three) stamps throughout its life, each represented by a different stamp style:

- **DEV Stamp Style.** Typically, this stamp style uses a simple *Build Number* stamp for development builds, often using a source-control version number or just counts successive builds: i.e., dev-500, dev-501, dev-502; or simply 100, 101, 102, etc. See [Stamping](#).

Though stamps can be used as the basis of version control baseline, label or tag, that is strictly optional.

- **QA Stamp Style.** When a build has been deployed to quality assurance, the QA stamp style will typically contain a stamp such as qa-100, qa-101, qa-102, etc., identifying that the build has reached the quality assurance stage of the life-cycle. See [Applying Multiple Stamps to a Build](#).

- **PROD Stamp Style.** Will usually contain a different number (or identifier) that's easier for customer's to understand: i.e., 1.1, 1.2, 1.3. This stamp style may correspond with your release numbers, but it does not have to. See [Applying Multiple Stamps to a Build](#).

Once a stamp style is created, AnthillPro ties the stamp to the workflow, and automatically identifies all successive builds. When multiple stamp styles are used, AnthillPro will assign a new stamp to a build based on criteria you define when configuring the stamp style or based on the Create Stamp job step of the secondary workflow. See [Applying Multiple Stamps to a Build](#).

1. To create a new Stamp Style, select the **Stamp Styles** link and click **Create New**.
2. Give the Stamp Style a **name** and **description**.
3. Click **Save**.

Artifact Sets

In AnthillPro, an Artifact Set is a label for a collection of build artifacts. AnthillPro allows you to create as many Sets as you want, and then associate the build artifacts with a particular Artifact Set. This gives you fine-grained control over how the artifacts are used.

Some types of commonly used Artifact Sets:

- A **code library** used by multiple projects.
- A platform-specific **client**.
- **Images** or **video** incorporated into a product.
- **Documentation** of the shared library to be rolled into developer documentation (JavaDoc).
- The **deployable executable** of a top-level project.

In general, similar projects often have similar artifacts. In AnthillPro, instead of managing these artifacts on every project, you manage them as part of the Life-Cycle Model by creating Artifact Sets. When you create an Artifact Set on a Life-Cycle Model, that Artifact Set will be available to every project that uses the Model. Once that is done, you then associate the build artifacts with an Artifact Set during workflow configuration. For example, projects that generate a shared library might have an Artifact Set called `lib` which contains the library. On the build workflow, you associate the library with the `lib` Artifact Set. This makes the artifacts (the library) available to be used in dependencies, deployments, or other processes.

In AnthillPro, Artifact Sets are used to define dependency relationships: Any project that another project depends on generates one or more collections of files used by the dependent project. These collection of files are called an Artifact Set in AnthillPro. When configuring a dependency, you select which Artifact Sets a project depends on (see [Dependency Management](#) for more information on how AnthillPro manages dependencies). Artifact Sets are also used to specify files that may be used in secondary workflows. The most common secondary workflow that uses Artifact Sets is the deployment workflow: AnthillPro will send the artifacts to a different location (say a server in QA) based on the Artifact Sets you configure (see [Setting Up a Deployment Process](#) for more information on how Artifact Sets are used in deployments).

1. To create a new Artifact Set, select the **Artifact Sets** link and click **Create New**.

2. Give the Artifact Set a **name** and **description (optional)**.
3. Click **Save**.
4. See also Artifact Set Security and Setting an Artifact Retention Policy (under Cleanup).

Artifact Set Security

Once an artifact set has been configured, it is possible to secure it. This will enable you to control, based on the default permissions for user roles, who can download/resolve an artifact set and who can set security permissions for an artifact set. If you do not see a security icon (a yellow badge) in the Operations menu, Artifact Set Security has not been enabled or you do not have the appropriate permission.

- Please see Securing Artifact Sets for detailed instructions. To properly secure an artifact set, you will need to first enable default permissions, change a system setting (**System > Server Settings > Security**) and then set the artifact security for the artifact set(s) by selecting the Security icon on the Operations menu.

Cleanup

The Cleanup configures AnthillPro to periodically clean up old Build Lives, build requests, and miscellaneous jobs generated by a project. Cleanup is on a per-project basis, so every project that uses the same Life-Cycle Model will have the same policy.

If you set a cleanup policy that keeps the 5 most recent successful Build Lives for 1 week, AnthillPro will enforce the policy on a per-workflow basis. So, if you have a "trunk" and a "branch" originating workflow in the same project, AnthillPro will keep the 5 most recent successful Build Lives of "trunk" AND of "branch" for 1 week.

There are three basic options for Build Life cleanups:

- **Delete.** Fully deletes the Build Life, artifacts and logs from the server. See Setting an Artifact Retention Policy for more.
- **Inactivate.** Deletes the Build Life's artifacts and marks the Build Life as inactive, preventing execution of workflows on the Build life and any use of the inactive Build Life to satisfy dependencies. See Setting an Artifact Retention Policy for more.
- **Archive.** Deletes the Build Life's artifacts and marks the Build Life with an archived status. The Build Life can later be unarchived and its artifacts rebuilt with an unarchive process. See Setting an Artifact Retention Policy for more.

Click the Cleanup link to determine a Cleanup Schedule, choose how Builds Requests are handled, and set how Build Lives are cleaned-up.

Setting an Artifact Retention Policy

There is no one-size-fits-all "policy": what you want to keep and for how long depends on your organizational guidelines/processes. Eventually, you will need to decide how long to keep the artifacts in order to save disk space. Following is a general guideline that should work for most organizations:

- **CI/Daily Builds (Success):** 15 days: the 5 most recent builds marked as "success" (for CI and/or nightly builds. Any build sent to testing, generally released requires different retention policy).
- **Any Build (Failure):** 3 days / 0 builds for failed. No reason to keep them -- often failures do not result in artifacts.
- **Builds Sent for Testing:** 30 days for testing builds (or the minimum testing cycle, whichever is longer). Most Agile testing cycles are shorted than 1-month, so keeping the artifacts around for at least that long is a good idea. If your testing cycle is longer than 30 days, then keeping the artifacts until the testing cycle is complete is a good idea.
- **General Release Builds:** Production should be kept around until the build has reached its end of life or regulations allow for deletion.

See also Artifact Sets above.

Cleanup Schedule

Cleanup is kicked-off via a schedule, which must be created at **System > Schedules** under the Project Support menu (see Create Schedules). AnthillPro will automatically execute the cleanup rules you set in the Cleanup Build Requests and Cleanup Build Lives sections, based on the cleanup-policy schedule. In addition, the cleanup policy allows you to configure cleanups of lockable resources, operational workflows and operational jobs.

1. Go to **System > Cleanup** under the Project Support menu.
2. Click the **Edit** icon and configure the cleanup policy:
 - **Schedule.** Select the schedule. If you do not have an appropriate schedule, please create one before continuing. See Create Schedules.

How often the schedule should fire is determined, in part, by the cleanup rules set on the Life-cycle Model. For example, if you have cleanup run on a weekly schedule and the *Cleanup Build Request* set to **Never**, no build requests will be cleaned up. Likewise, if you have *Cleanup Build Lives* for "All Build Lives" set to be kept for 2 weeks, the first time the cleanup schedule runs (one week from the starting date) no Build Lives will be deleted because the policy only allows Build Lives to be cleaned up every 2 weeks. However, the second time (two weeks from the initial creation of the schedule) the schedule runs all Build Lives 2 weeks and older will be cleaned up.

- **Active.** Check the box to activate the cleanup schedule. Make sure the schedule you selected above is also marked as Active. To do this go to **System > Schedules** and ensure that it says "Yes" under the Active menu. If you select an inactive schedule, your cleanup policy will not run, even if it is marked as active here.
- **Days to Keep Lockable Resources.** Give the number of days that lockable resources should be kept after they are last used. This applies to limited resources (e.g., working directories). If you are using unique resources (such as unique working directories), this setting can come in handy to save space. Default value is 30 days. Use -1 to never clean up.
- **Days to Keep Operation Workflows.** Give the number of days you want to keep operational workflow executions. Use -1 (the default value) to never clean up. If you rarely use operational projects, the default values should suffice. The only cleanup option is to delete the workflow.
- **Days to Keep Operation Misc Jobs.** Give the number of days that operational miscellaneous jobs, such as cleanup jobs and notification jobs, are kept for. Default value is 1 day. Use -1 to never clean up. If you rarely use operational projects, the default values should suffice. The only option is to delete the jobs.

3. Click **Done**.

- See Cleanup Build Requests and Cleanup Build Lives.

Cleanup Build Request


Use this section to determine when **Build Requests** are cleaned up. Once set, the Cleanup Schedule will apply these settings every time it automatically fires. For example, if your cleanup schedule fires once a week, but you set the Misc. Job field to 100 days, when the cleanup schedule fires, only jobs that are older than 100 days will be cleaned up.

- To schedule the automatic deletion of build requests, select the **Edit** icon of the **Cleanup Build Request** menu.
- In the **Misc. Jobs** field, enter the **number of days** a miscellaneous job will exist before it is cleaned up. To keep jobs indefinitely, leave this field blank.
- In the **Build Requests Without Build Life** field, enter the **number of days** a build request that did not produce a build life will exist. To keep all requests, leave this field blank. Click **Save**.

Cleanup Build Lives

The cleanup of Build Lives is based on the status the Build Life has achieved. For example, set the cleanup to keep all Build Lives a minimum number of days or choose a specific status, such as *failure*, to expire after a minimum number of days, etc. Once set, the Cleanup Schedule will apply these settings every time it automatically fires. For example, if your cleanup schedule fires once a week, but you set the All Build Lives field to 100 days, when the cleanup schedule fires, only Build Lives that are older than 100 days will be cleaned up. To set the Build Life cleanup policy:

- Under the **Cleanup Build Lives** menu, select the **Edit** icon for the appropriate status.

Cleanup Build Lives:				
Action	Status	Keep Days	Keep Latest	Type
	All Build Lives	365	4	Delete

- Keep Days.** Give the minimum number of days to keep all Build Lives that have been assigned this status (leave the field blank to keep all Build Lives).

Keep Days:

Use this field to set a minimum number of days which to keep all buildlives before they expire.

(blank for all buildlives never expire)

For example, setting the days at "5" will keep every Build Life that has been assigned this status for a minimum of 5 days, unless overridden by the Keep Latest Build Life specification. When the cleanup schedule fires, it will clean up Build Lives that are older than 5 days.

- Keep Latest.** Specify the number of the most recent Build Lives that have been assigned this status from being

cleaned up, regardless of the Keep Days policy.

Keep Latest:

Use this field to prevent the latest X buildlives from being cleaned up.

For example, if the Keep Days field is set to "5" (days) and the Keep Latest field set to "2" (Latest Build Lives), when the 2 most recent Build Lives that have been assigned this status are 5 days old, they will not be cleaned up by the Keep Days policy. This allows you to keep a copy of the most recent Build Lives no matter how old they are.

4. **Cleanup Type.** Select the cleanup type from the drop-down menu. Options are Delete (see Delete a Build Life), Inactivate (see also Inactivate Build Life), and Archive (see also Archive and Unarchive Build Life).

Cleanup Type:

The type of cleanup to be performed.

5. Click **Save**.
6. Repeat Items One thru Five for every status.

Editing a Life-Cycle Model

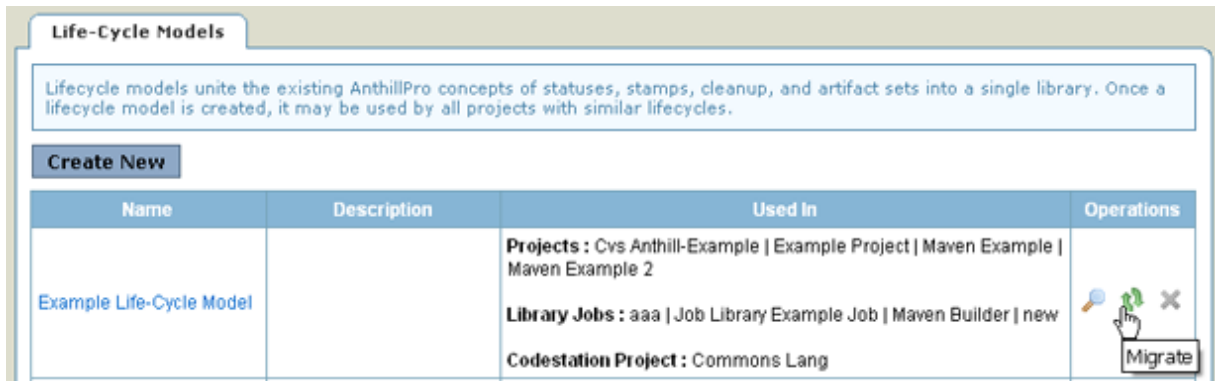
Edit a Life-Cycle Model any time by following the procedures outlined in the Creating a Life-Cycle Model section of this tutorial. Simply select the Model to edit and click the **Edit** icon for the feature to be change. Click **Save**.

- Projects using the same Life-Cycle Model may not all respond the same to changes. Each project may need to be reconfigured if any changes are made to its Life-Cycle Model.

Migrating a Life-Cycle Model

Migrating a Life-Cycle Model will transfer all Projects, Library Workflows and Library Jobs using it. The Status, Artifact Sets, and Stamps will be reconfigured during the migration.

1. Go to **System > Life-Cycle Model** under the Project Support menu.
2. On the **Life-Cycle Models** page, click the **Migrate** icon of the Life-Cycle Model you want to migrate.



3. **Migrate To.** Select the Life-Cycle Model to **migrate to** from the drop-down menu and click **Migrate**.

4. On the Migrate Life-Cycle Model, select:

- **Status Migration.** Select one or more new statuses to migrate each of the old statuses to.
- **Artifact Migration.** Select a new artifact set to migrate each of the old artifact sets to.

The Life-Cycle Model being migrated to must have artifact sets configured that AnthillPro can map to. See Artifact Sets.

- **Stamp Migration.** Select a new stamp style to migrate each of the old stamp styles to.

The Life-Cycle Model being migrated to must have stamps configured that AnthillPro can map to. See Stamp Style.

5. Click **Migrate** then **Done**.

Life-Cycle Model Security

User access to a Life-Cycle Model is managed on the **Security** tab. Administrators can define what roles have access to read, write, or determine security for Life-Cycle Models. You need administrative permissions to set environment security. See Manage Security.

1. Go to **System > Life-Cycle Models** under the Project Support menu.
2. On the **Life-Cycle Models** page, select the appropriate Life-Cycle Model from the list.
3. Select the Life-Cycle Model's **Security** tab and click **Edit**.
4. Check the **Roles** and **permissions** for this agent. See Define Roles and Set and Manage Permissions.
5. Click **Save** then **Done**.

Chapter 38. Notifications

AnthillPro sends a number of notifications to users in a variety of formats. Most commonly, AnthillPro is configured to send either an e-mail or IM message regarding the state of a CI build. The recipient list of these notifications, as with other notifications AnthillPro sends out, are usually tied to the LDAP integration, etc., which is configured as part of AnthillPro's security system; as well as your mail server and/or IM provider. This allows you an easy way to send notifications to AnthillPro users (it is also possible to send notifications to non-users as well). You can even compose custom notifications that send the information you want in the format you want.

In addition to sending notifications via e-mail and IM, administrators can send out announcements to all AnthillPro users. Once an announcement is sent, it will appear at the top of the UI for all AnthillPro users. Common uses for announcements are to let everyone know when the server is going down for maintenance, report network problems, etc.

Configure Mail Server

AnthillPro requires an external mail server to route messages through (via SMTP). It is recommended that you set up an 'AnthillPro' user on the server so that e-mails can be sent (e.g., anthill@yourcompany.com).

- Some e-mail servers and firewalls will treat e-mails with different **Sender** and **User** names as suspicious. If this happens, use the same name for both fields.

Mail Server Prerequisites

- You must have administrative permissions to the System page. See Manage Security.
- An external mail server must already have an 'AnthillPro' user account created (e.g., anthill@yourcompany.com).

Mail Server

1. Go to **System** > **Mail Server** from the Notification menu.
2. Click **Edit** and configure the Mail Server.
 - **Name** the Mail Server.
 - **Mail Host.** Give the mail server host name. This should be an IP or a network name that can be resolved from the server running Anthill Pro.
 - **Mail Port.** Provide the mail server port. Default value is 25.
 - **Sender.** Give the email address to send messages as (e.g., anthill3@yourcompany.com). This will appear in the 'From' header of the e-mail.
 - **User Name.** Provide the user name to be used to authenticate with the SMTP server. The user name is used to log into the SMTP server and to actually send the e-mail.
 - **Password.** Give the password to be used to authenticate with the SMTP server.
 - **Use TLS if available.** Check the box to use TLS.

3. Click **Set**.

Configure Instant Messaging

AnthillPro can be used to send instant messages using Google Talk, Jabber, or MSN IM. For example, AnthillPro may be configured to send an instant message to a group of committing developers when a build fails, etc. See Managing Notifications.

The Google Talk and Jabber integrations are configured by following the XMPP IM link under the Notification menu on the System page. The integration can be disabled any time by clicking the **Disable** button. Using the **Test User ID** and **Service Status** fields allow you to troubleshoot the connection.

Google Talk

AnthillPro requires an external Google Talk server to route messages. Any steps within AnthillPro relying on Google Talk (XMPP) will not work until this configuration is complete.

Google Talk Prerequisites

- You must have administrative permissions to the System page. See Manage Security.
- AnthillPro notification and security systems must be configured in order for the integration to work. See Managing Notifications.
- AnthillPro is added as a Google Talk user.

Configure Google Talk

1. Go to **System** > **XMPP IM** from the Notification menu.
2. On the **XMPP** page, click **Enable** then click **Edit**.
3. Configure the integration:
 - **Name.** Give the name of this XMPP configuration.
 - **Server Host.** Provide the XMPP server host name, typically an IP or host name that can be resolved from the server running AnthillPro. For example: jabber.org or talk.google.com.
 - **Server Port.** Give the XMPP server port. If using multiple ports, separate them with commas (e.g., 5222, 5223, 5224).
 - **Domain.** Provide the XMPP user domain.
 - **Username.** Give the user name (e.g., anthill3).
 - **Password.** Give the password.
4. Click **Set** then **Done**.
5. To begin sending instant messages, click **Enable**. (If **Disable** is visible, the instant message integration is currently active.)

Check Google Talk Connection

If there is a problem with the Google Talk integration, first check that the **Service Status** field says connected. Troubleshoot the connection by giving a **Test User ID** and clicking the Test button. If there is a problem, AnthillPro will give you a message (see below).

We could not connect to the XMPP Server

Test User ID:

Test

Service Status:

Not Connected

SASL authentication failed

Jabber

AnthillPro requires an external Jabber server to route messages. Any steps within AnthillPro relying on Jabber (XMPP) will not work until this configuration is complete.

Jabber Prerequisites

- You must have administrative permissions to the System page. See Manage Security.
- AnthillPro notification and security systems must be configured in order for the integration to work. See Managing Notifications.
- AnthillPro is added as a valid Jabber user.

Configure Jabber

1. Go to **System** > **XMPP IM** from the Notification menu.
2. On the **XMPP** page, click **Enable** then click **Edit**.
3. Configure the integration:
 - **Name.** Give the name of this XMPP configuration.
 - **Server Host.** Provide the XMPP server host name, typically an IP or host name that can be resolved from the server running AnthillPro. For example: jabber.org or talk.google.com.
 - **Server Port.** Give the XMPP server port. If using multiple ports, separate them with commas (e.g., 5222, 5223, 5224).
 - **Domain.** Provide the XMPP user domain.
 - **Username.** Give the user name (e.g., anthill3).
 - **Password.** Give the password.
4. Click **Set** then **Done**.
5. To begin sending instant messages, click **Enable**. (If Disable is visible, the instant message integration is cur-

rently active.)

Test Jabber Connection

If there is a problem with the Jabber integration, first check that the **Service Status** field says connected. Trouble shoot the connection by giving a **Test User ID** and clicking the Test button. If there is a problem, AnthillPro will give you a message (see below).

Test User ID:	<input type="text" value="anthill3"/>	<input type="button" value="Test"/>
Service Status:	Not Connected	Could not connect to jabber.urbancode.com:5222.

MSN IM

Configure MSN instant messaging. AnthillPro requires an external IM server to route messages. Any steps within AnthillPro relying on MSN IM will not work until it is configured.

MSN IM Prerequisites

- You must have administrative permissions to the System page. See Manage Security.
- AnthillPro notification and security systems must be configured in order for the integration to work. See Managing Notifications.
- AnthillPro is added as a valid MSN IM user.

Configure MSN IM

1. Go to **System > MSN IM** from the Notification menu.
2. On the **MSN** page, click **Enable** then click **Edit**.
3. Configure the integration:
 - **Name.** Give the name of this MSN configuration.
 - **Email (owner).** Provide the e-mail address used to authenticate with the MSN server.
 - **Password.** Give the password.
4. Click **Set** then **Done**.
5. To begin sending instant messages, click **Enable**. (If Disable is visible, the instant message integration is currently active.)

Test MSN IM Connection

If there is a problem with the MSN IM integration, first check that the **Service Status** field says connected. Trouble shoot the connection by giving a **Test User ID** and clicking the Test button. If there is a problem, AnthillPro will give you a message (see below).

Unable to connect to server/send message: null

Test User ID:

ahp3@yourcompany.com

Test 

Service Status:

Not Connected

Setting Up Notifications

Once configured, AnthillPro can routinely run builds with no interaction with the development or build team. However, if it fails to communicate the results of builds, deployments, and promotions back to the team, it is only useful when people log-in to the system to manually check the status. That is not very good. A better model is to have AnthillPro send e-mails, instant messages, and other notifications to select team members.

The fundamental unit for managing notifications sent by AnthillPro is the Notification Scheme. A notification scheme sets rules determining what groups of users are sent which kind of notification about specified events. Each workflow is configured with a notification scheme within AnthillPro. The same scheme may be shared by many workflows, even workflows in different projects.

A Notification Scheme is composed of the following (See Composing a Notification Scheme):

- **Recipient Generator.** Selects which users to contact. For existing mailing lists (e.g., generated from LDAP), it may be helpful to create a user representing a mailing list. See Manage Security.
- **Event Selector.** Selects which events (created by a workflow event) to send notifications about.
- **Medium.** Selects how the notification will be sent (e-mail or instant message).
- **Notification Template.** AnthillPro uses Velocity templates to generate the notification text. See Scripting Notification Templates.

Notification Prerequisites

- In order to manage notifications, you must have read and write permissions to the System page.
- An existing Authentication Realm and at least one user must be created. See Manage Security.
- In order to send e-mails, the email server must be configured. See Configure Mail Server.
- To send instant messages, AnthillPro must first be configured. See Configuring Instant Messaging.

Recipient Generator

Users in the system are going to be interested in different events, with most users only interested in notifications about projects they work on. For example, developers are more concerned about the state of a broken build than a system administrator, who is more interested in knowing when a build does not start due to problems with source control.

The default options, **All Users**, **All Users for Project**, **Committing Developers**, or **Tasked Users** allow administrators to determine who is notified based on the Role(s) assigned to them. Adding additional Roles (see Manage Se-

curity) or creating new Generator scripts (see also Recipient Generator Scripts) provides flexibility to customize Recipient Generators.

1. Go to **System > Recipient Generators** under the Notification menu.
2. Click the **Create Recipient Generator** button.
3. Select the **Notification Recipient Generator Type** from the drop-down menu.
 - **Fixed.** Enter the recipient information such as email or IM address. Proceed to Item Five.
 - **Role-Based.** Generate notification recipients based on roles (see Manage Security). Proceed to Item Six.
 - **Scripted.** Use a BeanShell script to generate notification recipients (see Recipient Generator Scripts). Proceed to Item Seven.
4. Click **Set**.
5. Configure the **Fixed Notification Recipient Generator**.
 - **Name** the Generator.
 - **Description.** Provide a description.
 - Click **Save**. The Recipients menu will appear.
 - Under the **Recipients** menu, give the **Email, XMPP, and/or MSN** address(es). Click **Add**.

The e-mail values can be set as AnthillPro properties. This enables you to send a notification to an existing mail list, etc., without having to configure individual e-mail addresses in AnthillPro. For example, in the Email field, you can configure the selector to send a notification to: `${property:mailing-list}@mycompany.com`. Then add a Project Property called `mailing-list` with a value of `myProjectTeam`, to your project. Once this is done, AnthillPro will send notifications to `myProjectTeam@mycompany.com`. Using this approach, you can send notifications to any number of lists. See Managing Properties.

- Click **Done**.
6. Configure the **Role-Based Notification Recipient Generator**.
 - **Name** the Generator.
 - **Description.** Provide a description.
 - **Roles.** Check the Roles to associate with this Generator. See Manage Security.
 - Click **Save** then **Done**.
 7. Configure the **Scripted Recipient Generator**.
 - **Name** the Generator.
 - **Description.** Provide a description.
 - **BeanShell script.** Input the script that generates the list of recipients. See Recipient Generator Scripts.
 - Click **Save** then **Done**.

New Recipient Generator

A scripted recipient generator uses a beanshell script to generate notification recipients.

Name: * Scripted Recipient Generator The name of the recipient generator. *denotes required field

Description: Uses BeanShell script The description of the recipient generator.

Beanshell Script: * The script to generate the list of recipients.

```
1 import com.urbancode.anthill3.domain.security.*;
2 userFactory = UserFactory.getInstance();
3 result = new User[] {
4     userFactory.restoreForNameAndRealm("Eric", "Anthill"),
5     userFactory.restoreForNameAndRealm("dbeckham", "Anthill")
6 };
```

Save Cancel

Event Selector

In order to determine which events should trigger a notification, AnthillPro maintains a set of event selectors. These are scripts that examine events generated by workflows and workflow requests, and return true if a notification should occur and false otherwise. For most circumstances, users will not need to write new event selectors. The product ships with a good set of default scripts:

- **Workflow Success.** Triggers notifications when the workflow completed successfully.
- **Workflow Fails.** Triggers notifications when the workflow completes without success.
- **Workflow Success or Failure.** Triggers when a workflow completes regardless of status.
- **Build Request Failed.** Triggers when a workflow does not occur due to an error in the build request. See Scripting.

1. Go to **System > Event Selectors** under the Notification menu.
2. Click the **Create Event Selector** button on the main page.
3. **Name** the new Event Selector, give it a **description (optional)**, provide a **script**, and click **Save**. See Event Selector (Scripting).

WORKFLOW SUCCESS

An event selector listens to events in the system. It is responsible for identifying one type of event that triggers notifications. A notification scheme may use several event selectors.

*denotes required field

Name: * The name of the event selector.

Description: The description of the event selector.

Beanshell Script: The script returns true if we should notify users about this event.

```

1  import com.urbancode.anthill13.domain.workflow.*;
2
3  result = false;
4  if (event instanceof WorkflowEvent &&
5      event.getCase().isComplete() &&
6      event.getCase().getStatus().isSuccess()) {
7      result = true;
8  }
9  return result;
10
```

Notification Template

AnthillPro Notification Templates are Velocity templates that take information about the build and produce a document. Appropriate templates need to be paired with the appropriate event cases. If there is a `WorkflowEvent`, that means that a `WorkflowCase` was created and will be the main piece of information provided to the script as `workflow`. Likewise, a `BuildRequest` will be passed to the script as `request`.

Different templates will also be appropriate to send out on different mediums. Generally, a template used for instant messages will be very short, while one that targets e-mail will need to be longer. A template targeted at instant messages will not have a subject section.

AnthillPro ships with a catalog of Velocity templates that can be used as a reference in creating your own templates. At **System > Notification Templates** under the Notification menu, view the templates similar to the one you wish to create. See [Scripting Notification Templates](#).

- Best practices for writing templates is beyond the scope of this tutorial. However, it should be noted that the type of event handled will dictate the inputs to the template. See [Velocity Documentation](http://velocity.apache.org/) [http://velocity.apache.org/].

1. Go to **System > Notification Templates** under the Notification menu.
2. Click the **Create New** button.
3. Input a **name**, **description (optional)**, **context script (optional for IM)**, and **template script**. Click **Set**. See [Scripting Notification Templates](#).

Composing a Notification Scheme

Once the notification components have been configured, they are composed into a new Notification Scheme. The Notification Scheme defines who will be notified and when they will be notified. For example, a "Default" scheme can be used to notify everyone on a project about a failed build; or a "Unit Test" scheme can be configured to notify

only committing developers if the tests fail, etc.

To compose a scheme, a series of Recipient Generators (e.g., one Generator for sending e-mails; one copying [CC] e-mails; and for sending blind [BCC] e-mails) are typically created. In addition, the appropriate Case Selector(s) must be created for each scheme. See Recipient Generator and Event Selector.

1. Go to **System > Notification Schemes** under the Notification menu.
2. Click the **Create Notification Scheme** button on the **Notification Schemes** main page.
3. **Name**, provide a **description (optional)**, and click **Save**.
4. Click the **Add Who-When** button to specify who should be contacted when a specific event occurs.
5. Specify **combinations of users, message types, and events** that need unique notification formats.
 - **Recipient Generator.** Select the Recipient Generator from the drop-down menu. This field allows AnthillPro to send an e-mail "To" users.
 - **CC Recipient Generator.** Select the appropriate Recipient Generator from the drop-down menu. This field allows AnthillPro to copy, "CC", a defined group on an e-mail. Leave this field blank to not send copies or if configuring Instant Messaging.
 - **BCC Recipient Generator.** Select the appropriate Recipient Generator from the drop-down menu. This field allows AnthillPro to anonymously copy, "BCC", a defined group on an e-mail. Leave this field blank to not send copies or if configuring Instant Messaging.
 - **Case Selector.** Select the appropriate case selector from the drop-down menu.
6. Click **Save**.
7. Determine how users are notified. Click the **Add Medium-Template** button.
 - Select the **Notification Medium and the Message Template** from the drop-down menus. Click **Save**.

Creating a series of who-when-how combinations establishes a scheme of sending the appropriate notifications out that can be used between project teams.

Setting Up and Using Announcements

Once an announcement is created by an AnthillPro administrator, the message will automatically appear at the top of the AnthillPro UI for every user. Users can hide any announcement by following the hide link within any announcement. Selecting the announcements link in the upper right-hand corner of the AnthillPro UI allows users to view all active messages, including those previously hidden.

Creating Announcements

1. Go to **System > Announcements** under the Server menu.
2. On the **Announcements** page, Click the **New** button.
3. Configure the message:
 - **Message.** Enter the message. Up to 4,000 characters maximum.

- **Priority.** Select the priority (default is normal) from the drop-down menu.
 - **Low.** Displays message in gray.
 - **Normal.** Displays message in black.
 - **High.** Displays message in red.
- Click **Save** then **Done**.

Editing and Deleting Announcements

1. Go to **System > Announcements** under the Server menu.
2. On the **Announcements** page, select the **Edit** icon of the announcement to be edited.
3. Modify the text in the **Message** field and/or change the **Priority**. When done, click **Save** then **Done**.
 - If an announcement has been hidden by a user, changes are only viewable on the Your Announcements page. See Viewing Announcements.
4. To delete an announcement, click the **Delete** icon and then **OK**.

Viewing Announcements

Once an announcement has been sent, it appears at the top of every user's AnthillPro UI. If an announcement has been hidden, it can be viewed on the Your Announcements page.

- Hide any announcement by following the **hide** link within any announcement. If the hidden announcement is edited by the sender, it will not automatically reappear.



- Select the **announcements** link in the upper right-hand corner of the AnthillPro UI to view all active announcements on the **Your Announcements** page.

A screenshot of the "Your Announcements" page. It shows a table with two columns: "Created" and "Message".

Created	Message
2008-04-02 01:03	The source configuration system has suffered a critical failure. Watch this banner for updates when the server has been restored.
2008-04-02 12:36	On April 2, all build servers will be shut down between 9:00 PM and 11:00 PM for maintenance.

Chapter 39. Properties

AnthillPro properties use the syntax `${property:[property_name]}` to pass property values to commands being executed by AnthillPro. Often, properties are used to manage variables passed into commands, agent filters, and custom stamping algorithm templates.

For example, if a workflow has a property named `platform` passed into a Nant script via the Nant integration, something similar to `PlatProp=${property:platform}` would be set on the Nant Properties tab. At the command line, this would translate to the flag `-D:PlatProp:x86` if the value of the property was 'x86'. Here, the property is used as if it was passed to the Nant executable on the command line.

Properties are resolved from the smallest scope to the most general scope (e.g., Job properties are resolved before project properties, which in turn are resolved before system properties).

Property Resolution Order:

1. **Job Instance and Job Iteration Property.** Configured in the settings of an iterating job, they set the parameters for a single job that is run many times. See Job Instance and Job Iteration Properties.
2. **Build Request Property.** When set as a workflow property as shown below, the property will get pushed to the build request before the build begins. See Build Request Properties.
3. **Workflow Property.** Used to specify a property for a given execution of a workflow. Workflow properties are also used to send a build to a particular platform when writing native code for multiple platforms, etc. See Workflow Properties. See also:
 - Workflow Request Properties
 - Workflow Properties (from Originating-build Workflow)
4. **Build Life Property.** See Build Life Properties.
5. **Build Life Originating Workflow Request Property.** When set as a workflow property, the value will get pushed to the Build Life originating workflow before the build begins. See Build Life Originating Workflow Request Properties.
6. **Project Environment Property.** Used to customize deployments based on the environment they are deployed to. Project Environment properties are automatically placed as environment variables for all commands run in the target environment. See Project Environment Properties.
7. **Project Property.** Used for all workflows regardless of the target environment. See Project Properties.
8. **Environment Property.** Environment properties are used to set default values for a particular environment. Other property types, except system properties, always override the environment-level properties See Environment Properties.
9. **System Property.** Used to set default values for a particular property for all workflows and projects system wide. See System Properties.

In addition, AnthillPro has two other property types, which are resolved separately from those listed above. See Specialized Builder Properties and Agent Properties.

Property Resolution

Below is a general outline of property resolution used throughout the AnthillPro system:

Property	Resolution
<code>#{property:<property-name>}</code> <i>or</i> <code>#{p:<property-name>}</code>	Return the non-agent property value if defined. If not defined, return the given expression.
<code>#{property-:<property-name>}</code> <i>or</i> <code>#{p-:<property-name>}</code>	Return the non-agent property value if defined. If not defined, return null.
<code>#{property?:<property-name>}</code> <i>or</i> <code>#{p?:<property-name>}</code>	If defined as a non-agent property, replace with the value. If not defined, replace the expression source.
<code>#{agent:<property-name>}</code> <i>or</i> <code>#{a:<property-name>}</code>	Return the agent property value if defined. If not defined, return the given expression.
<code>#{agent?:<property-name>}</code> <i>or</i> <code>#{a?:<property-name>}</code> <i>or</i> <code>#{?:<property-name>}</code>	If defined as an agent property, replace with the value. If not defined, replace the expression source.
<code>#{<property-name>}</code> <i>or</i> <code>#{<property-name>}</code>	Return the agent property value if defined. If not defined, return null.

Empty Values in Properties

Empty-value properties enable users to set a standard for the available attributes a user can provide when executing a resource. For example, using an empty-value property (for both user-overrideable and non-overrideable properties) will allow the same library job to be reused across a large number of projects, even if different projects require different values. If most projects using the same library job need to pass parameters at run time, but some of them don't, using an empty-value property gives users the option to define a required value or simply pass a blank value that AnthillPro will effectively ignore. Used in this way, empty-value properties can be used to create standardized templates for creating new projects.

Example usage for resolving empty properties using a simple echo use case. If nothing follows "echo," that means empty has been returned.

Property	Resolution
<code>echo #{property:property}</code>	When "property" is not defined:

	<p>echo \${property:property}</p> <p>When "property" is defined on project as "value":</p> <p>echo value</p> <p>When "property" is defined on project as "":</p> <p>echo</p> <p>When "property" is defined on agent as "value":</p> <p>echo \${property:property}</p>
<p>echo \${p:property}</p>	<p>When "property" is not defined:</p> <p>echo \${p:property}</p> <p>When "property" is defined on project as "value":</p> <p>echo value</p> <p>When "property" is defined on project as "":</p> <p>echo</p> <p>When "property" is defined on agent as "value":</p> <p>echo \${p:property}</p>
<p>echo \${property?:property}</p>	<p>When "property" is not defined:</p> <p>echo</p> <p>When "property" is defined on project as "value":</p> <p>echo value</p> <p>When "property" is defined on project as "":</p> <p>echo</p> <p>When "property" is defined on agent as "value":</p> <p>echo</p>
<p>echo \${p?:property}</p>	<p>When "property" is not defined:</p> <p>echo</p> <p>When "property" is defined on project as "value":</p> <p>echo value</p> <p>When "property" is defined on project as "":</p> <p>echo</p>

	<p>When "property" is defined on agent as "value": echo</p>
echo \${agent:property}	<p>When "property" is not defined: echo \${agent:property}</p> <p>When "property" is defined on project as "value": echo \${agent:property}</p> <p>When "property" is defined on project as "": echo \${agent:property}</p> <p>When "property" is defined on agent as "value": echo value</p>
echo \${a:property}	<p>When "property" is not defined: echo \${a:property}</p> <p>When "property" is defined on project as "value": echo \${a:property}</p> <p>When "property" is defined on project as "": echo \${a:property}</p> <p>When "property" is defined on agent as "value": echo value</p>
echo \${agent?:property}	<p>When "property" is not defined: echo</p> <p>When "property" is defined on project as "value": echo</p> <p>When "property" is defined on project as "": echo</p> <p>When "property" is defined on agent as "value": echo value</p>
echo \${a?:property}	<p>When "property" is not defined: echo</p> <p>When "property" is defined on project as "value":</p>

	<p>echo</p> <p><i>When "property" is defined on project as "":</i></p> <p>echo</p> <p><i>When "property" is defined on agent as "value":</i></p> <p>echo value</p>
<p>echo \${?:property}</p>	<p>When "property" is not defined:</p> <p>echo</p> <p>When "property" is defined on project as "value":</p> <p>echo</p> <p>When "property" is defined on project as "":</p> <p>echo</p> <p>When "property" is defined on agent as "value":</p> <p>echo value</p>
<p>echo \${property}</p>	<p>When "property" is not defined:</p> <p>echo \${property}</p> <p>When "property" is defined on project as "value":</p> <p>echo \${property}</p> <p>When "property" is defined on project as "":</p> <p>echo \${property}</p> <p>When "property" is defined on agent as "value":</p> <p>echo value</p>
<p>echo \${bsh:ProjectLookup.getCurrent().getName() }</p>	<p>When "property" is not defined:</p> <p>echo Resolve to Null Test</p> <p>When "property" is defined on project as "value":</p> <p>echo Resolve to Null Test</p> <p>When "property" is defined on project as "":</p> <p>echo Resolve to Null Test</p> <p>When "property" is defined on agent as "value":</p> <p>echo Resolve to Null Test</p>

<pre>echo \${bsh:PropertyLookup.get("property")}</pre>	<p>When "property" is not defined:</p> <pre>echo \${bsh:PropertyLookup.get("property")}</pre> <p>When "property" is defined on project as "value":</p> <pre>echo value</pre> <p>When "property" is defined on project as "":</p> <pre>echo</pre> <p>When "property" is defined on agent as "value":</p> <pre>echo \${bsh:PropertyLookup.get("property")}</pre>
--	--

Setting Default Properties

Properties are commonly used to enforce conventions across the system, environment, project, workflow, and/or job. To control how a group of related resources act, a property is set at the lower property-resolution level: For example, to set a default property for every workflow within a project, you would set a project property. Likewise, if most projects require a specific tool when deploying, then setting a system property will help ensure that all projects deploy correctly. For the odd projects that don't conform to the default patterns, the property may be overridden at a level closer to the job execution (i.e., if you have a system default property, it may be overridden at the project, workflow, or job levels on a as-needed basis).

Consider this common scenario: A team stores most of their build scripts for almost every project in a folder called "build" under the root of their project. There are a number of ways (and places) to configure a property that can act as a default for most projects, ensuring the correct script is used when building: Basic AnthillPro configuration suggests creating either a project or workflow property called `build.dir` and reference the property in the configuration that kicks off the build script(s). This works for all but the corner cases, but can include a lot of configuration if you have a lot of projects and/or workflows. For setting these types of standards, however, it's usually easier to create a system level property (**System > Properties**) and only specify the `build.dir` property on the odd projects/workflows that don't conform to the common pattern. In this case, the System level property would be "build". If an absolute path is needed for some reason, System Properties can reference runtime properties. So a System Property of `${property:work.dir.path}/build` would result in the absolute path to the same build sub-folder. A workflow that set the `build.dir` to `/some/other/path` would use that -- completely ignoring the System level setting because properties set closer to the execution of a step always override properties set at higher levels.

Job Instance and Job Iteration Properties

Job iteration properties are configured after iterating a job. They are typically used to set the parameters for a single job that is run many times, with a slightly different parameter each time. For example, a job iteration property might include variations such as "module to compile", "test to run", etc.

Typically, properties set on the job iteration are used by a scripted agent selection (filter) script that you must create yourself. The script evaluates the property when the job is requested, and then runs each iteration on the agent(s) that have the appropriate (i.e., corresponding) property-value pair. For example, if you set a property called "my_property" with a value of "1" on iteration one, the agent selection script will only run iteration one on the agent(s) that have "my_property" with a value of "1" set on it/them. (More at **Tools > anthill3-dev-kit.zip > Script-**

ing > Example: Select Agents Based on Job-iteration Properties).

Job Iteration Properties Prerequisites

- You must have AnthillPro administrative privileges to set job properties. See Manage Security.
- A Project with an originating workflow and at least one job must already be active.
- The job must already be iterated. See Iterate a Job.
- If you are planning on using job-iteration properties to determine which agent each iteration will run on, you will need to create a new Agent Filter Script. See **Tools > anthill3-dev-kit.zip > Scripting > Example: Select Agents Based on Job-iteration Properties**.
- Corresponding properties must be set on the agent(s) so that when the agent filter script runs, it will select the correct agent(s). See Configure and Edit Agent Filters.

Setting Job Iteration Properties

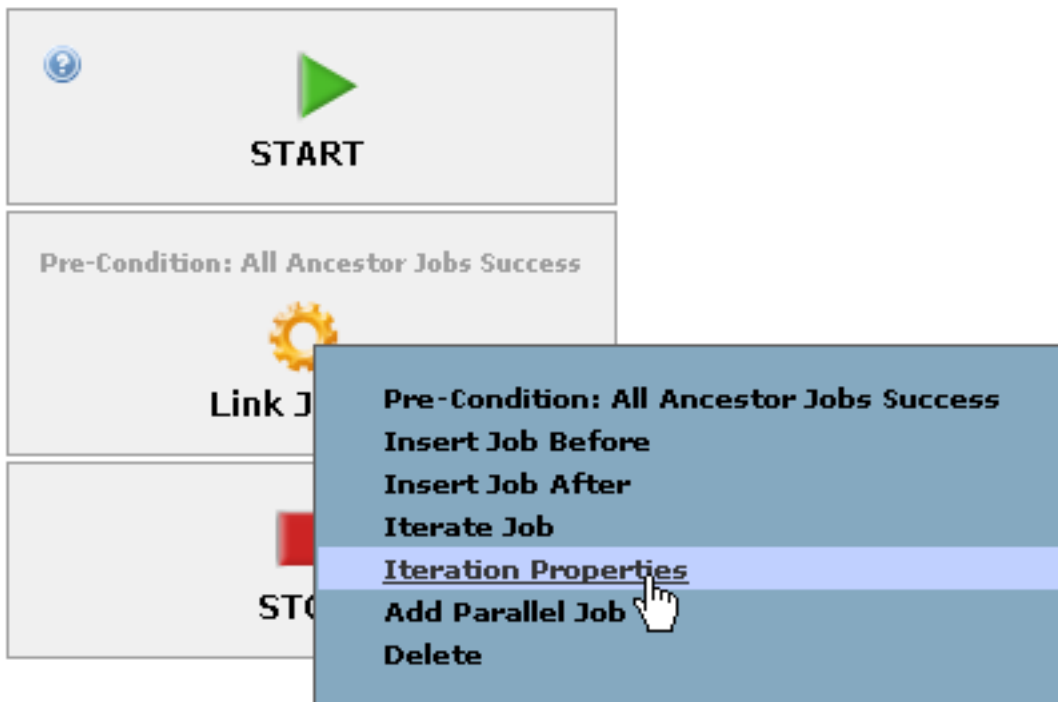
Properties are set in the *name=value* format. For example, to iterate a job to run two different test suites, you can set a property (with a name of `dir-location`) that has two values (`/temp/test1` and `/temp/test2`), with each value corresponding to a single iteration of a job:

Iteration Name: +		dir-location	✘
Iteration 1	✘	/temp/test1	
Iteration 2	✘	/temp/test2	

Once you configure the names and properties of each job iteration, AnthillPro will set the iteration name and iteration number as job properties during each job iteration. The iteration name is set as: `anthill.job.iteration.name` and the iteration number is set as: `anthill.job.iteration`. Once the job has run, you can view this on the Build Life.

In addition, corresponding properties must be set on the appropriate agent(s) if you want AnthillPro to select an agent based on job-iteration properties. See Configure and Edit Agent Filters.

1. See **Tools > anthill3-dev-kit.zip > Scripting > Example: Select Agents Based on Job-iteration Properties** before continuing.
2. Go to **Administration** and select the **workflow** that contains the iterated job. See Iterate a Job.
3. On the workflow page, select the **Definition** tab.
4. Click the iterated job and select **Iteration Properties** from the menu.



5. **Iteration Properties page.** The iterations you already set (see Iterate a Job) are populated with a default name (e.g., Iteration 1 , Iteration 2, etc.). If you want to change the name of an iteration, click the name and type the new name.

To add a new property name, give the name of the iteration property (e.g., dir-location) and click **Set**. (Note you may have to scroll to the bottom of the page, depending on how large the browser window is.) When that is done, you can create iteration values (e.g., /temp/test1 and /temp/test2) for each property by clicking the appropriate cell and typing.

You can also include a value when creating the property name. When created in this way, the same value (e.g., /temp/test1), will be added to each iteration property. So, if you have a number of iterations that require the same property value, you only have to type it once if you configure the property in this manner.

6. Click **Save**.
7. When done, click **Cancel** to exit the Iteration Properties page.

Build Request Properties

When set as a workflow property as shown below, the property will get pushed to the build request before the build begins.

Example Build Request Property:

```
import com.urbancode.anthill3.runtime.scripting.helpers.*;
import com.urbancode.anthill3.domain.buildrequest.*;

//look up the current build request

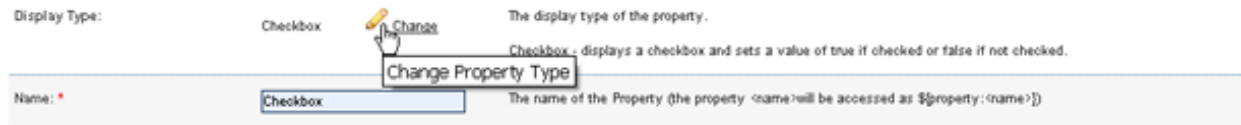
BuildRequest br = BuildRequestLookup.getCurrent();
```

```
//set the property
br.setProperty("myProperty", "myValue");
```

Workflow Properties

Workflow properties are used to specify a property for a given execution of a workflow (see Managing Properties). A common example of a workflow property is the tag (or label, or baseline) used in a "Build From Label" workflow that checks out a particular version of the source code from source control and performs a build.

Workflow properties are also used to send a build to a particular platform when writing native code for multiple platforms, etc. Workflow properties may be either optional or required, may have default values, may be locked so that end users can't change the default, or dynamic (i.e., AnthillPro generates the property value based on user-defined criteria). Once a workflow property is created, the display type may be changed by selecting the **Change** icon on the configuration page.



Additionally, use the Operations menu (on the workflow's Properties tab) to (a.) determine the order of properties; (b.) edit a configured property; and (c.) delete a property.

There are six different workflow property types to choose from. They range from the simple Checkbox type to the Text Area type, which can pass small scripts:

- **Checkbox.** Use to require a user to check the box in order to run a workflow.
- **Multi-Select.** Use to set pre-defined values where multiple values can be selected.
- **Select.** Use to present a selection of pre-defined values where one value can be selected.
- **Text.** Use to present a text input where a value can be entered. The Text property type may be scripted.
- **Text (secure).** Use to present a text input where a secure value must be entered. The value is obfuscated in any output.
- **Text Area.** Use to present a text area input where a longer value can be entered. The Text Area property type may be scripted.

See also Using Workflow Properties for Build Life Notes and Cascading Workflow Properties.

Workflow Properties Prerequisites

- You must have AnthillPro administrative privileges. See Manage Security.
- A Project with an originating workflow and at least one job must already be active.

Using Checkbox Workflow Properties

Use the checkbox workflow property to require a user to check the box in order to run a workflow. The Checkbox

property displays a checkbox and sets a value of true if checked or false if not checked. By default, the property is selected; however the administrator may choose not to have the property set as a default. To run the workflow without the property, the administrator must allow users to override the default. For example, the user can run the workflow even though no value is selected. See [Setting Properties](#).

1. Go to **Administration** and select the **workflow** to add properties to.
2. On the workflow page, select the **Properties** tab.
3. Click **New Property**, select **Checkbox** form the drop-down menu, click **Select**, and provide the following:
 - **Name.** Give the name of the property: the property <name> will be accessed as `${property:<name>}`.
 - **Description.** Provide an optional description for this property. If given, the description will be shown when prompting users for a value.
4. **Value.** Select the value type. Choose either Defined (see Item 5), Job Execution (see Item 6), or Scripted (see Item 7).
5. **Defined-value property.** Basic property type. Displays a checkbox for the user when running a build. To configure, give the following:
 - **Default Value.** Check the box to make this a default value. If unchecked, and the **User Can Enter Value** option is configured, the user can run the workflow without checking the box. When the default value is set but the user can't override the value, the property will not be visible to the user.
 - **User Can Enter Value.** To allow users to specify a non-default value when running this workflow, check the box. If the user may override the default property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name (defined above) as the Label.
 - Click **Save** and go to Item 8.
6. **Job-execution property.** Use this property type if the value, default value, or options are to be generated by a job that executes on an agent.
 - **Agent Filter.** Select the agent filter script to be used when determining the value of this property. See [Agent Filter \(Selection\) Scripts](#).
 - **Job.** Select the job to execute that will set the property value using this property name. Either project or library jobs may be used. When this job runs, the value of this property will be set.
 - **User Can Enter Value.** To allow users to enter a value when running this workflow, check the box. If the user may enter a value for the property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name (defined above) as the Label.
 - Click **Save** and go to Item 8.
7. **Scripted property.** Scripted value properties are properties where the value, default value, or options are generated by a script that is executed before workflow execution.
 - **Value Script.** Give a BeanShell script that returns *Boolean*. The BeanShell script will be passed the following implicit variables when applicable: project, workflow, Build Life, environment.
 - **User Can Enter Value.** To allow users to enter a value when running this workflow, check the box. If the user

may enter a value for the property, give the following:

- **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name (defined above) as the Label.
- Click **Save** and go to Item 8.

8. To use Workflow properties run a build.

Using Multi-select Workflow Properties

Use the multi-select workflow property to set pre-defined values where multiple values can be selected. Once set, all selected values will become values of the property in one comma-separated value. See Setting Properties.

1. Go to **Administration** and select the **workflow** to add properties to.
2. On the workflow page, select the **Properties** tab.
3. Click **New Property**, select **Multi-select** from the drop-down menu, click **Select**, and provide the following:
 - **Name.** Give the name of the property: the property <name> will be accessed as `${property:<name>}`.
 - **Description.** Provide an optional description for this property. If given, the description will be shown when prompting users for a value.
4. **Value.** Select the value type. Choose either Defined (see Item 5), Job Execution (see Item 6), or Scripted (see Item 7).
5. **Defined-value property.** Basic property type. Displays options for the user (either the property name or default value) when running a build. To configure, give the following:
 - **Default Value.** Check the box to make this a default value. *The default value must be a value from the allowed values list configured below if more than one value will be allowed.*
 - **User Can Enter Value.** To allow users to enter a new value when running this workflow, check the box. If the user may override the property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.
 - **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.
 - **Allowed Values.** Give the values users will select from when running the workflow. Each value must be input on a separate line. If any value is given, you must also define a default value above.
 - Click **Save** and go to Item 8.
6. **Job-execution property.** Use this property type if the value, default value or options are to be generated by a job that executes on a agent.
 - **Agent Filter.** Select the agent filter script to be used when determining the value of this property. See Agent Filter (Selection) Scripts.
 - **Job.** Select the job to execute that will set the property value using this property name. Either project or library jobs may be used. When this job runs, the value of this property will be set.

- **User Can Enter Value.** To allow users to enter a value when running this workflow, check the box. If the user may enter a value for the property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.
 - **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.
 - Click **Save** and go to Item 8.
7. **Scripted property.** Scripted value properties are properties where the value, default value or options are generated by a script that is executed before workflow execution.
- **Value Script.** Give a BeanShell script that returns *string array or collection*. The BeanShell script will be passed the following implicit variables when applicable: project, workflow, Build Life, environment.
 - **User Can Enter Value.** To allow users to enter a value when running this workflow, check the box. If the user may enter a value for the property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.
 - **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.
 - Click **Save** and go to Item 8.
8. To use Workflow properties run a build.

Using Select Workflow Properties

Use the select workflow property to present a selection of pre-defined values where one value can be selected. See Setting Properties.

1. Go to **Administration** and select the **workflow** to add properties to.
2. On the workflow page, select the **Properties** tab.
3. Click **New Property**, select **Select** from the drop-down menu, click **Select**, and provide the following:
 - **Name.** Give the name of the property: the property <name> will be accessed as `${property:<name>}`.
 - **Description.** Provide an optional description for this property. If given, the description will be shown when prompting users for a value.
4. **Value.** Select the value type. Choose either Defined (see Item 5), Job Execution (see Item 6), or Scripted (see Item 7).
5. **Defined-value property.** Basic property type. Displays choices for the user (either the property name or default value) when running a build. To configure, give the following:
 - **Default Value.** Check the box to make this a default value. *The default value must be a value from the allowed values list configured below if more than one value will be allowed.*
 - **User Can Enter Value.** To allow users to specify a new value when running this workflow, check the box. If the user may override the property, give the following:

- **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.
 - **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.
 - **Allowed Values.** Give the values users may provide for this property. Each value must be input on a separate line.
 - Click **Save** and go to Item 8.
6. **Job-execution property.** Use this property type if the value, default value or options are to be generated by a job that executes on an agent.
- **Agent Filter.** Select the agent filter script to be used when determining the value of this property. See Agent Filter (Selection) Scripts.
 - **Job.** Select the job to execute that will set the property value using this property name. Either project or library jobs may be used. When this job runs, the value of this property will be set.
 - **User Can Enter Value.** To allow users to enter a value when running this workflow, check the box. If the user may enter a value for the property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.
 - **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.
 - Click **Save** and go to Item 8.
7. **Scripted property.** Scripted value properties are properties where the value, default value or options are generated by a script that is executed before workflow execution.
- **Value Script.** Give a BeanShell script that returns *string array or collection*. The BeanShell script will be passed the following implicit variables when applicable: project, workflow, Build Life, environment.
 - **User Can Enter Value.** To allow users to enter a value when running this workflow, check the box. If the user may enter a value for the property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.
 - **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.
 - Click **Save** and go to Item 8.
8. To use Workflow properties run a build.

Using Text Workflow Properties

Use the text workflow property to present a text input where a value can be entered. The Text property type may be scripted. See Scripting and Setting Properties.

1. Go to **Administration** and select the **workflow** to add properties to.
2. On the workflow page, select the **Properties** tab.

3. Click **New Property**, select **Text** from the drop-down menu, click **Select**, and provide the following:
 - **Name.** Give the name of the property: the property <name> will be accessed as `${property:<name>}`.
 - **Description.** Provide an optional description for this property. If given, the description will be shown when prompting users for a value.
4. **Value.** Select the value type. Choose either Defined (see Item 5), Job Execution (see Item 6), or Scripted (see Item 7).
5. **Defined-value property.** Basic property type. Displays a text input for the user (either the property name or default value) when running a build. To configure, give the following:
 - **Default Value.** Check the box to make this a default value. *The default value must be a value from the allowed values list configured below if more than one value will be allowed.*
 - **User Can Enter Value.** To allow users to specify a new value when running this workflow, check the box. If the user may override the property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.
 - **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.
 - Click **Save** and go to Item 8.
6. **Job-execution property.** Use this property type if the value, default value or options are to be generated by a job that executes on an agent.
 - **Agent Filter.** Select the agent filter script to be used when determining the value of this property. See Agent Filter (Selection) Scripts.
 - **Job.** Select the job to execute that will set the property value using this property name. Either project or library jobs may be used. When this job runs, the value of this property will be set.
 - **User Can Enter Value.** To allow users to enter a value when running this workflow, check the box. If the user may enter a value for the property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.
 - **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.
 - Click **Save** and go to Item 8.
7. **Scripted property.** Scripted value properties are properties where the value, default value or options are generated by a script that is executed before workflow execution.
 - **Value Script.** Give a BeanShell script that returns *a string value*. The BeanShell script will be passed the following implicit variables when applicable: project, workflow, Build Life, environment.
 - **User Can Enter Value.** To allow users to enter a value when running this workflow, check the box. If the user may enter a value for the property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.
 - **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.

- Click **Save** and go to Item 8.

8. To use Workflow properties run a build.

See also Using Workflow Properties for Build Life Notes.

Using Text (secure) Workflow Properties

Use to present a text input where a secure value must be entered. The value is obfuscated in any output (i.e., all strings matching secure value are replaced with **** in the output). The Text (secure) property type may be scripted. See Scripting and Setting Properties.

1. Go to **Administration** and select the **workflow** to add properties to.
2. On the workflow page, select the **Properties** tab.
3. Click **New Property**, select **Text-secure** from the drop-down menu, click **Select**, and provide the following:
 - **Name.** Give the name of the property: the property <name> will be accessed as `${property:<name>}`.
 - **Description.** Provide an optional description for this property. If given, the description will be shown when prompting users for a value.
4. **Value.** Select the value type. Choose either Defined (see Item 5), Job Execution (see Item 6), or Scripted (see Item 7).
5. **Defined-value property.** Basic property type. Displays a secured field for the user (either the property name or default value) when running a build. To configure, give the following:
 - **Default Value.** Check the box to make this a default value. If unchecked, the user must manually check the value when executing the workflow.
 - **Confirm.** Re-enter the secure value.
 - **User Can Enter Value.** To allow users to specify a new value when running this workflow, check the box. If the user may override the property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.
 - **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.
 - Click **Save** and go to Item 8.
6. **Job-execution property.** Use this property type if the value, default value or options are to be generated by a job that executes on an agent.
 - **Agent Filter.** Select the agent filter script to be used when determining the value of this property. See Agent Filter (Selection) Scripts.
 - **Job.** Select the job to execute that will set the property value using this property name. Either project or library jobs may be used. When this job runs, the value of this property will be set.
 - **User Can Enter Value.** To allow users to enter a value when running this workflow, check the box. If the user may enter a value for the property, give the following:

- **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.
 - **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.
 - Click **Save** and go to Item 8.
7. **Scripted property.** Scripted value properties are properties where the value, default value or options are generated by a script that is executed before workflow execution.
- **Value Script.** Give a BeanShell script that returns *a string value*. The BeanShell script will be passed the following implicit variables when applicable: project, workflow, Build Life, environment.
 - **User Can Enter Value.** To allow users to enter a value when running this workflow, check the box. If the user may enter a value for the property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.
 - **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.
 - Click **Save** and go to Item 8.
8. To use Workflow properties run a build.

Using Text Area Workflow Properties

Use the text area workflow property to present a text area input where a longer value can be entered. The Text Area property type may be scripted. See Scripting and Setting Properties.

1. Go to **Administration** and select the **workflow** to add properties to.
2. On the workflow page, select the **Properties** tab.
3. Click **New Property**, select **Text Area** from the drop-down menu, click **Select**, and provide the following:
 - **Name.** Give the name of the property: the property <name> will be accessed as `${property:<name>}`.
 - **Description.** Provide an optional description for this property. If given, the description will be shown when prompting users for a value.
4. **Value.** Select the value type. Choose either Defined (see Item 5), Job Execution (see Item 6), or Scripted (see Item 7).
5. **Defined-value property.** Basic property type. Displays a text area for the user (either the property name or default value) when running a build. To configure, give the following:
 - **Default Value.** Check the box to make this a default value. *The default value must be a value from the allowed values list configured below if more than one value will be allowed.*
 - **User Can Enter Value.** To allow users to specify a new value when running this workflow, check the box. If the user may override the property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.

- **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.
 - Click **Save** and go to Item 8.
6. **Job-execution property.** Use this property type if the value, default value or options are to be generated by a job that executes on an agent.
- **Agent Filter.** Select the agent filter script to be used when determining the value of this property. See Agent Filter (Selection) Scripts.
 - **Job.** Select the job to execute that will set the property value using this property name. Either project or library jobs may be used. When this job runs, the value of this property will be set.
 - **User Can Enter Value.** To allow users to enter a value when running this workflow, check the box. If the user may enter a value for the property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.
 - **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.
 - Click **Save** and go to Item 8.
7. **Scripted property.** Scripted value properties are properties where the value, default value or options are generated by a script that is executed before workflow execution.
- **Value Script.** Give a BeanShell script that returns *string value*. The BeanShell script will be passed the following implicit variables when applicable: project, workflow, Build Life, environment.
 - **User Can Enter Value.** To allow users to enter a value when running this workflow, check the box. If the user may enter a value for the property, give the following:
 - **Label.** Give a label for this property to be shown when prompting users for value. Leave blank to use the Name as the Label.
 - **Is Required.** Check the box to require the user to select a value for this property when the workflow is run.
 - Click **Save** and go to Item 8.
8. To use Workflow properties run a build.

Workflow Request Properties

Workflow request properties are configured when a user adds input fields to a workflow (showing on the dashboard for build workflows and the secondary process pop-up for secondaries). For example, a workflow request property could be "clean vs. incremental build" or "Branch to build from". See Setting Properties.

Workflow Properties (from Originating-build Workflow)

The workflow property (from originating-build workflow) is similar to the Workflow Request Property; however, it is a secondary process resolving the variables from a previous Workflow. See Setting Properties.

Cascading Workflow Properties

If you have properties set on a workflow (e.g., parent workflow) that are also required when kicking off a child workflow (i.e., a deployment workflow), you can configure your job to pass the configured properties to the child workflow. Cascading properties is commonly accomplished when using the Run Another Workflow step. To cascade properties when kicking off another workflow:

1. Go to your job and select the **Insert After** icon of the step immediately preceding where the **Run Another Workflow** step is to be included. Typically, this step is added near the end of a job.
2. Open the Miscellaneous folder, select **Run Another Workflow**, and click **Select**.
3. Configure step:
 - **Name** the step.
 - **Description.** Give an optional description.
 - **Workflow.** Select the workflow you want AnthillPro to run. Only secondary workflows available to this project will appear in the drop-down. Click **Set**.
 - **Environment.** Select the environment the child workflow is going to run in. If you don't see the environment you want, that is because the workflow you selected does not participate in that environment.
 - **Wait for Workflow.** Checking the box will force AnthillPro to wait for the parent workflow to complete before kicking off the child workflow. If checked, this step will not run if any previous job steps fail. Note that while waiting for the parent workflow to complete, the child workflow holds any Lockable Resource and counts as running on an agent.
 - **Pass Properties.** Select one of the options to cascade the properties from the parent workflow to the child workflow.
 - *Do not pass properties.* This is AnthillPro's default behavior. No request properties defined on the parent workflow will cascade to the child workflow.
 - *Only properties with matching names.* A request property on the parent workflow will cascade to the child if and only if the child workflow defines a property of the same name. For example, if both the parent workflow and the child workflow are configured with a property named 'agents', that property (including the value set on the parent workflow) will be passed to the child when this step executes. However, if the parent workflow has a property named 'Linux' configured but the child workflow does not, the 'Linux' property will not cascade to the child workflow.
 - *Pass all properties.* Pass all request properties from the current originating workflow to the child workflow, even if no matching property name is configured on the child workflow. For example, if both the parent workflow and the child workflow are configured with a property named 'agents', but have different values, the value set on the parent workflow will be used when running the child workflow. If a particular property is configured on the parent workflow, but not on the child, that property -- including its value -- will be passed to the child workflow at run time.
 - **Show Additional Options (optional; advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.

- **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
- **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

4. Click **Save**.

Dynamic Workflow Properties

The exercise below shows how dynamic workflow properties can be used to control which agents will be targeted for deployment. Configuration consists of creating a scripted multi-select workflow property, a scripted agent filter, and adding the run another workflow step to a job.

1. Go to your build job (originating workflow) and set the iterations to -1 and unique agents. When set to -1, AnthillPro will iterate a job equal to the number of agents returned by the agent selection filter (discussed below). Selecting Unique Agents will cause AnthillPro to run each iteration on a separate agent.
2. Create a new Agent Filter script (see Configure and Edit Agent Filters for more) that will return all agents that are referenced in the property value discussed below. The property name in the example is 'agents'. The property name in this script should be the name of the workflow property you will add in subsequent steps. Here is the script:

```
import com.urbancode.anthill3.domain.agent.Agent;
import java.util.*;

return new Where() {
    public Agent[] filter(Agent[] agents) {
        agentNames = PropertyLookup.get("agents");
        nameArray = agentNames.split(",");
        agentList = new ArrayList();
        for (int a=0; a<agents.length; a++) {
            for (int n=0; n<nameArray.length; n++) {
                if (agents[a].getName().equals(nameArray[n])) {
                    agentList.add(agents[a]);
                }
            }
        }
        agents = agentList.toArray(new Agent[agentList.size()]);
        return agents;
    }
};
```

3. On your workflow, create a scripted multi-select property. In the example, It is named 'agents'. The Agent Filter script created above must use the same name. When configuring the property, it should be user overridable and required. The script for value is:

```
import com.urbancode.anthill3.services.agent.AgentManager;
import com.urbancode.anthill3.domain.agent.AgentFactory;

endpoints = AgentManager.getInstance().getOnLineEndpointArray(environment);
values = new String[endpoints.length];
for (int i=0; i<endpoints.length; i++) {
    agent = AgentFactory.getInstance().restoreByEndpoint(endpoints[i]);
    values[i] = agent.getName();
}
```

```
}  
return values;
```

4. Save your settings.

The next time the build runs, the user will be prompted to select an agent that was dynamically selected. The workflow will only be run if the user selects an agent from the list.

Build Life Properties

Set Build Life properties on originating workflows using an Evaluate Script job step. Build Life properties are typically used to hold variable data for builds (such as compiler version, etc.) or to create an audit trail. Once the property is set, and the build has run, the Build Life property will be visible on the Build Life page.

To use a Build Life property, add the Evaluate Script step to the job -- typically after the Populate Workspace and Get Change Log steps of the typical job. Once set on the originating workflow, Build Life properties are available to all secondary workflows (such as deployments, etc.) run on the Build Life, and are unique to the Build Life.

- For **originating workflows**, use: `BuildLifeLookup.getCurrent().setProperty("name", "value")` to set the property at build time.
- For **secondary workflows**, use: `${bsh: BuildLifeLookup.getCurrent().get(name)}` to resolve the property when the workflow is run.

For more information on AnthillPro scripting, see Scripting Basics.

Build Life Originating Workflow Request Property

When set as a workflow property, the value will get pushed to the Build Life originating workflow before the build begins. This can be useful if you set user-overrideable properties on the build workflow. See Workflow Properties to set the property.

Project-Environment Properties

Project environment properties may be used to customize deployments based on the environment they are deployed to. Project Environment properties are automatically placed as environment variables for all commands run in the target environment. This eases their use in shell scripts, Perl scripts, and make commands. They can also be looked up using the `${property:..}` syntax (see Scripting). They are set on the properties tab on the Administration page.

In a J2EE project, for example, a typical deployment might involve deploying an EAR file to a server with a specific environment, user name, and password. These properties may be marked as secured. If secured, the value of the properties will appear in the UI and logs as '*****', and the properties will be encrypted in the database. The secured option guards against the leak of information (such as database passwords) to those permitted to configure a project or read a project log, but who are not permitted to deploy to some environments.

Most projects participate in multiple environments. Because each environment is different, project environment properties must be set for each environment the project is associated with. The process is the same for each environment; however, depending on the environment the property may be different. See Setting Properties.

Project Environment Properties Prerequisites

- You must have AnthillPro administrative privileges. See Manage Security.
- A Project with an originating workflow and at least one job must already be active.

Set Project Environment Properties

1. Go to **Administration** and select the **project** to add properties to.
2. On the project page, select the **Properties** tab.

Use the **ALL** sub-tab to set properties for this project (see Project Properties). Any property set here will be set in every environment this project participates in.

The environments this project participates in are listed as sub-tabs. To set an environment-specific property, select an environment. These properties are typically unique to the environment.

3. Select a specific environment. Click the **Add Property** link on the Project Properties page. If the project is part of multiple environments, each environment will be listed separately. Set the project environment property for one environment then proceed to the others.
4. Set property:
 - **Name.** Give the name of the property: the property <name> will be accessed as `${property:<name>}`.
 - **Value.** Give the value, often a number or other value used to manage variables passed into commands, agent filters, and custom algorithm templates. See also Empty Values in Properties.
 - **Secure.** If using password properties and secure properties, check Yes. If the value does not need to be secure, check No. The value will be obfuscated in output and the UI.
 - **Pass to Builders.** Select Yes to pass this property to the builder. For example, if using Ant, AnthillPro typically calls the build.xml file for a build. However, if you select Yes, you can use AnthillPro to pass a property to the build script. The call would look something like `ant build.xml -Dproperty`. See Specialized Builder Properties.
 - **Set in Environment.** Select Yes to automatically set this property in the environment. This will allow the property to be passed to SCM steps and any other step that uses a shell command.
5. To add another property, select the **Add Property** link again. Click **Save** when done.
6. Repeat Items 3-5 to set project environment properties for other environments.

Project Properties

Project properties may be used for all workflows regardless of the target environment. They are set on the Properties tab on the Project's Administration page, and may be looked up by integration points and e-mail notifiers.

Project Properties Prerequisites

- You must have AnthillPro administrative privileges. See Manage Security.

- A Project with an originating workflow and at least one job must already be active.

Using Project Properties

1. Go to **Administration** and select the **project** to add properties to.
2. On the project page, select the **Properties** tab.
3. Select the **ALL** sub-tab. Click the **Add Property** link on the Project Properties page. Any property set here will be set in every environment this project participates in.

If the project is part of multiple environments, each environment will be listed separately. Set the project environment property for one environment then proceed to the others. See Project Environment Properties.

4. Set property:
 - **Name.** Give the name of the property: the property <name> will be accessed as `${property:<name>}`.
 - **Value.** Give the value, often a number or other value used to manage variables passed into commands, agent filters, and custom algorithm templates. See also Empty Values in Properties.
 - **Secure.** If using password properties and secure properties, check Yes. If the value does not need to be secure, check No. The value will be obfuscated in output and the UI.
 - **Pass to Builders.** Select Yes to pass this property to the builder. For example, if using Ant, AnthillPro typically calls the build.xml file for a build. However, if you select Yes, you can use AnthillPro to pass a property to the build script. The call would look something like `ant build.xml -Dproperty`. See Specialized Builder Properties.
 - **Set in Environment.** Select Yes to automatically set this property in the environment. This will allow the property to be passed to SCM steps and any other step that uses a shell command.
5. To add another property, select the **Add Property** link again. Click **Save** when done.

Environment Property

Environment properties are used to set default values for a particular environment. Other property types, except system properties, always override the environment-level properties. To set an environment property:

1. Go to **Agents > Environment**.
2. On the Environments Page, select the environment you want to set a property for.
3. Click **Add Property**.
4. Configure settings:
 - **Name** the property.
 - **Value.** Give the value of the property.
 - **Description.** Give an optional description.
5. Click **Done**.

System Properties

System properties are used to set default values for a particular property for all workflows and projects system wide. A frequently used default property name is useful when setting values for multiple projects. Other property types always override the system-level properties.

1. Go to **System > Properties** under the Server menu.
2. Click the **Add Property** button.
3. Configure settings:
 - **Name** the property.
 - **Value.** Give the value of the property. A frequently used default property name is useful when setting values for multiple projects. If using a server property, all other property types may override the server property. See also Empty Values in Properties.
 - **Description.** Give an optional description.
4. Click **Done**.

Specialized Builder Properties

Specialized 'builder' steps for tools like Ant, Maven, and Nant have a properties tab (these are properties as those tools understand them) available through the AnthillPro UI. Properties are then passed into the tools at the command line, using the tools specific format (usually `-D. . .`). Values for the properties may be one of the other property types or even a one-line BeanShell script.

Many users employ a builder property that looks up a job's stamp and then pass the stamp to a version property. In BeanShell, it would look like this: `#{bsh:WorkspaceVersion.get() }`. For more scripting, see **Tools > Development Kit Bundle > Scripting**.

Agent Properties

Agent properties can be used when the behavior of a command needs to change depending on the particular server it is running on. Agent properties are also typically used to identify where build or testing tools are installed on a particular agent, etc.

To lookup an agent property in the configuration of a step, no special identifier need be used. Just use the syntax `#{[agent_variable]}`. For example, the Ant installation directory for version 1.7.2 of Ant might be looked up using `#{ANT_HOME_1_7_2}`. For more scripting, see **Tools > Development Kit Bundle > Scripting**.

The agent properties (go to **Agents > Agent > select an agent**) are stored on the server. When the agent starts up, it passes all of its properties to the server, which then stores it. If a property is added or changed, or the agent is upgraded, etc., the properties will be automatically updated on the server.

Note that there are four groupings of agent properties, but you can only configure **Agent Properties**. The other three categories are set by the AnthillPro system and are not configurable via the UI.

See Manage Agents, Configure Agent(s), and Setting Properties.

AnthillPro Properties

AnthillPro has a number of properties automatically set by the system. A number of the properties are used in conjunction with integrations, and are automatically set during configuration or at runtime. Most users will not need to modify the AnthillPro properties once they are set.

Agent Properties	Description
codestation.cache.dir	Location of the agent Codestation cache. Default is the agent user's home directory plus '.codestation'.
codestation.cache.timeToLive	Number of days to keep something in the cache. Default is never.
codestation.noCache	Control over agent Codestation caching.
codestation.secondaryCache.dir	Location of secondary cache. Default is not enabled.
codestation.suppress.bom	Control over whether .bom bill of materials files should be created when artifacts are resolved. Default is true.
lab.manager.<configuration>	Identifies an agent as the agent installed in the <configuration> on VMware Lab Manager.
Build Life Properties	Description
anthill3.baseline.info	Set by ClearCase UCM Get ChangeLog and Rebase steps.
maven.lastUpdated	Identifies when a Codestation Build Life was downloaded from an external Maven 2 repository.
maven.proxiedRepository	Identifies which external Maven 2 repository a Codestation Build Life was downloaded from.
maven.version	Identifies an AnthillPro Build Life as a version to Maven 2.
Build Request Property	Description
anthill.buildlife.label	Value applied by a SCM Label step.
Job Properties	Description
anthill.job.iteration	Number of the job iteration being executed. Only available to iterated jobs.
anthill.job.iteration.name	User-defined name of the job iteration being executed. Only available to iterated jobs.
git-revision	Revision retrieved during the Git Populate Workspace step.
mercurial-revision	Revision retrieved during the Mercurial Populate Workspace step.
synergySessionToken	Synergy session token set by all Synergy steps.
work.dir.path	Set/updated to the current working directory of the job on the agent.
workspace.date	Set to the actual workspace date of the Build Life (which is the date of the source used). Formatted 'yyyy-MM-dd hh:mm:ss.S a z'.
workspace.version	Latest Build Life stamp applied during the job
Maven Properties	Description
maven.artifactId	Identifies a Codestation project, AnthillPro project or AnthillPro workflow with the Maven 2 artifactId.
maven.groupId	Identifies a Codestation project, AnthillPro project or

	AnthillPro workflow with the Maven 2 groupId.
--	---

Chapter 40. Build Life Links

Use Build Life Links to pass the URL of resources outside of AnthillPro to the Build Life. Once configured as a job step, the Link is available on the Reports tab for every Build Life that is associated with the job. See Add Build Life Link to Job and View Build Life Link.

For example, Build Life Links can be used to link to the different environments a project is deployed to (such as stage or production); to a third-party system that requested the build or deployment; or to detailed testing reports.

Before you begin make sure:

- You have administrative permissions. See Manage Security.
- A Life-Cycle Based Project is active. *Build Life Links cannot be used with Operational Projects.*
- The URL of the resource to be linked to is available.

Add Build Life Links as a step to any job that generates a Build Life. Once configured, the link will be passed to every Build Life associated with the job (see View Build Life Link). This section only covers the steps necessary to adding a Build Life Link step to a job. For detailed job configuration instructions.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. **Add Build Life Link**. Select the **Insert After** icon of the step prior to the point where the Build Life Link step is to be included (the step can be placed anywhere in the job). Expand the **Publishers** folder, select the **Add Build Life Link** step, and click **Select**.
 - **Name** the link.
 - **Link Name**. Give the name to be used in the Build Life Reports Tab.
 - **Link Description**. Provide the optional description.
 - **URL Script**. Give the script that determines the link URL. The script, using the standard AnthillPro script helpers, should return the value of the URL. (The example returns the URL for all environments the project is deployed to.) See Scripting.
 - **Show Additional Options (advanced)**. Select the Show Additional Options link to configure more options.
 - **Is Active**. Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Continuation Condition**. Select the condition which must be met for the process to continue (all steps pass; previous step passed; any step failed; always; or never).
 - **Ignore Failures**. Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript**. Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout**. Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed

out and abort it.

5. Click **Save**.

Once the job is configured with a Build Life Link step (see Add Build Life Link to Job), the link is available under the Build Life Reports tab when the job has been run.

1. Build the project.
2. Go to **Dashboard** and select the appropriate **workflow**.
3. On the **workflow Main** page, select the new **Build Life**.
4. Select the **Reports** tab.
5. Follow the link under the **Published Links** menu.

Part VIII. Advanced Usage

Advanced usage tools provide you in-depth views and fine-grained control for Build Lives, workflows and jobs. In addition, you can change your profile to personalize the Dashboard:

- **Build Life Tools.** There are a number of advanced tools available through the UI to help you diagnose problems and manage Build Lives.
 - **Workflow Tools.** Workflow priorities allow you to determine the order in which workflows run. Priorities are set as part of the configuration process. Workflow Requests tools allow you to create request plans and view the context of a workflow's request.
 - **Job Tools.** Provide an opportunity to abort, suspend, or prioritize jobs under the **Actions** menu.
 - **User Views.** Each AnthillPro user has some control over what is displayed on the Dashboard. You can add/change your password; user-repository alias; first and last name; e-mail and IM addresses; time zone; and the number of dashboard rows displayed. See Change User Profile.
 - **System Tray Monitor.** The System Tray Monitor provides feedback directly to the desktop, without having to open or refresh a browser.
-

Chapter 41. Build Life Tools

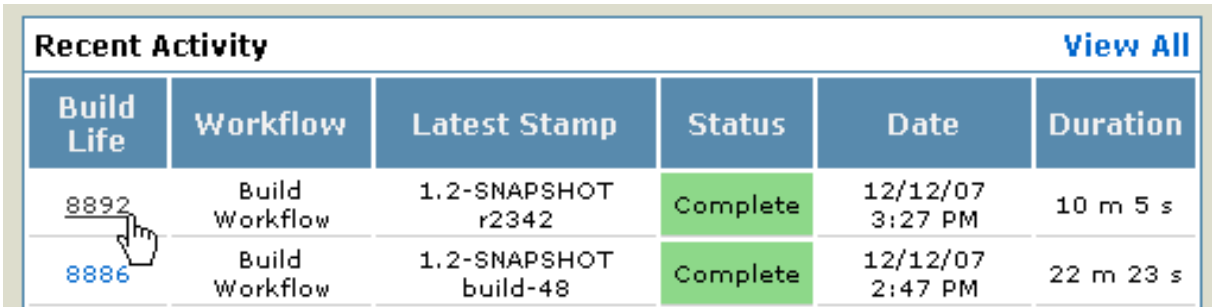
There are a number of advanced tools available through the UI to help you diagnose problems and manage Build Lives. All of the tools listed below pertain to individual Build Lives, and any actions you take will only effect or apply to the Build Life you are on.

Some of the Build Life tools can also be configured as part of the project's Life-Cycle Model. For example, deleting a Build Life and archiving a Build Life can be automated as part of the Life-Cycle Model's Cleanup Schedule. If you consistently use the manual versions of this feature, consider modifying the project's Life-Cycle Model.

Audit a Build Life


The Build Life page provides an overview of a Build Life history, showing the history of workflows executed for the Build Life, as well as what agents jobs ran on and when they ran.

1. To audit a **Build Life**, start from the **Dashboard** and select the **project** for which you want to audit a Build Life.
2. Select the **Build Life number** you are interested in.



Build Life	Workflow	Latest Stamp	Status	Date	Duration
8892	Build Workflow	1.2-SNAPSHOT r2342	Complete	12/12/07 3:27 PM	10 m 5 s
8886	Build Workflow	1.2-SNAPSHOT build-48	Complete	12/12/07 2:47 PM	22 m 23 s

3. Determine who triggered the workflow and how. Go to the **Build Life** page and click on the **Build Request** number.



Request	Workflow Name	ID	Start Date/Offset	Duration	Env/Selected Agent	Status	Actions
75603	Build Workflow	15137	2007-12-12 03:17:53 PM EST	00:10:05	build farm	Complete	
	Build	35513	00:00:00	00:10:05	cyprus	Success	

- In the example below, the user **etm** manually requested a build on December 2, 2007.



Request

Build Request 75603

Status: Created New Build Life

Date: Wed Dec 12 15:17:53 EST 2007

Requested By: etm (Manual)

Project: Maven Example

Workflow: Build Workflow

Log:

Agent: cyprus

Property Name	Value
No request properties	

4. **View similar logs to see who made the request to deploy a build to the QA environment** If this build was triggered by a schedule, that would be reflected here as well.

5. **View dependencies by selecting the Dependencies tab** The Dependency tab details what Build Lives a particular Build Life is using. It also details uses of this Build Life by other projects.

Depends On:

Project	Build Date	Build Life Id	Latest Status	Latest Stamp
Project B (SVN) - project b build workflow	2007-09-16 08:35:35 PM EDT	741	success	dev-27
Project C (SVN) - project c build workflow	2007-09-16 08:44:03 PM EDT	743	success	dev-16

In Use By:

Project	Build Date	Build Life	Latest Status	Latest Stamp
Not Currently In Use By Any Other Buildlives				

6. **Check status history and stamp** Status and stamp history displays the history of when the project attained each status or stamp. If it attained a certain status or stamp more than once, each time that happened it would show up here.

- Select the job number on the Status History or Stamp history to view the job trace.

Status Name	Workflow/Env/Job
success	Build Workflow - Build (35513)
failure	Build Workflow - Build (35513)
success	Build Workflow - Build (35513)

Stamp Value	Stamp Style	Workflow/Env/Job
1.2-SNAPSHOT r0	dev	Build Workflow - Build (35513)
1.2-SNAPSHOT r2342	dev	Build Workflow - Build (35513)

Inactivate a Build Life

Inactivating a Build Life removes the artifacts and permanently disables a Build Life. Once inactivated, a Build Life can never be enabled nor used in dependency relationships. (To temporarily deactivate a Build Life, see Archive and Unarchive Build Life.)

AnthillPro keeps a detailed record of all inactive Build Lives; however, they will not be visible on the normal project Dashboard pages. Inactive Build Lives are identified with a tombstone icon next to the Build Life number when a Search is run, and on the individual Build Life page. See Finding Inactive Build Lives.

Build Life

1342

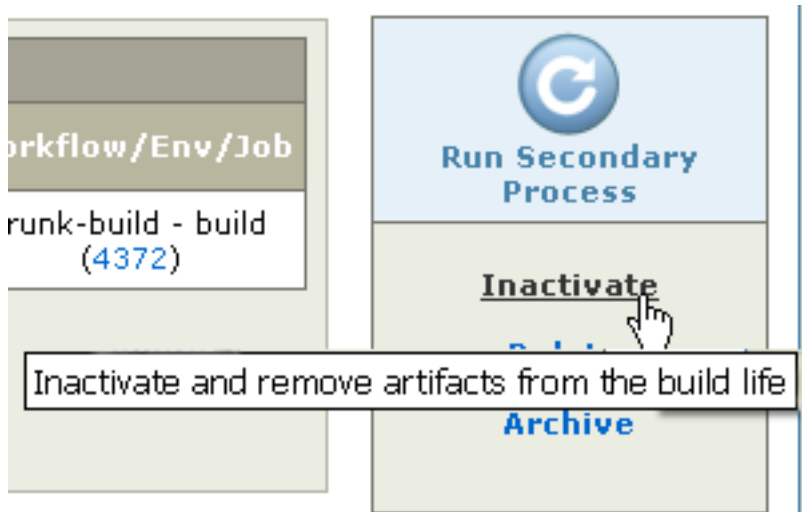
INACTIVE

Run Secondary Process

Delete

AnthillPro will not allow a Build Life to be inactivated until all the dependency relationships, etc., have been resolved. If you attempt to inactivate a Build Life that is in use, the warning "This BuildLife could not be deactivated. It is in use by active Build Lives: XXXX" will appear.

1. Go to **Dashboard** and select the **Workflow** associated with the Build Life to be inactivated.
2. On the **Main Workflow** page, select the appropriate **Build Life**.
3. On the **Main Build Life** page, select the **Inactivate** link.



4. Click **OK** in the dialogue box.

Inactive Build Lives do not appear on the regular Dashboard pages, but can be accessed using the Search page.

1. Go to **Search > Workflow**.
2. Select a Project from the drop-down menu.
3. Determine how many workflows will be displayed per page.
4. Click **Search**.

Inactivated Build Lives have a tombstone icon next to the number. Selecting the Build Life number links to the Main Build Life page.

Build Life | Build Request | **Workflow** | Status History | Task History | Misc Jobs

Search for Workflows using the following criteria:

Project:

Results Per Page: Number of results per page. Limited to 1000.

First (Showing page 47 of 48) Last
 ◀ 44 | 45 | 46 | 47 | 48 ▶

Build Life	Project	Workflow	Latest Stamp	Status	Date
49	XPetStore (Perforce)	DeployWorkflow	QA-54	Failed	1/9/07 11:13 AM
50	XPetStore (StarTeam)	DeployWorkflow	QA-54	Failed	1/9/07 11:07 AM
50	XPetStore (StarTeam)	Build Workflow	QA-54	Complete	1/9/07 11:04 AM

Archive and Unarchive Build Life

Archiving a Build Life stores the build properties (time of build, labels, build number, etc.) and removes the binary artifacts from AnthillPro's data warehouse. Once archived, any dependencies in the Build Life will be locked (see Archive Build Life). Archived Build Lives are automatically assigned the **Archived** status, and can be identified under the Status History menu on the Build Life Main page.

Status History	
Status Name	Workflow/Env/Job
Archived	Add Link - Add Link (2403)

Unarchiving a Build Life runs a build based on the stored information, reproduces the binary artifacts, and makes the Build Life available to normal AnthillPro processes. See Unarchive Build Life.

In order to archive a Build Life, the SCM/Repository must be labeled during the originating workflow build process. Otherwise, the warning "Unable to archive the build life because the originating workflow did not label the repository" will appear.

1. Go to **Dashboard** and select the **Workflow** associated with the Build Life to be archived.
2. On the **Main Workflow** page, select the appropriate **Build Life**.
3. On the **Main Build Life** page, select the **Archive** link.



4. Click **OK** in the dialogue box.

When unarchiving a Build Life, the stored properties are used to reproduce the binary artifacts and restore the Build Life to its original condition. Because dependencies generally do not need to be rebuilt, AnthillPro does not assume they were built against the same label. If you need to rebuild dependencies for an unarchived Build Life, force the dependencies to build when building the master project.

1. Go to **Dashboard** and select the **Workflow** associated with the Build Life to be unarchived.
2. On the **Main Workflow** page, select the appropriate **Build Life**.
3. On the **Main Build Life** page, select the **Unarchive** link.



4. Click **OK** in the dialogue box.

Delete a Build Life

Deleting a Build Life permanently removes all traces of the Build Life from AnthillPro, and cannot be reversed. To

temporarily deactivate a Build Life, see Archive and Unarchive Build Life. To permanently remove the artifacts from the Build Life but still keep a record of its existence, see Inactivate Build Life.

- You must have permissions to delete a Build Life. See Manage Security.
- Go to **Dashboard** and select the **Workflow** associated with the Build Life to be deleted.
 - On the **Main Workflow** page, select the appropriate **Build Life**.
 - On the **Main Build Life** page, select the **Delete** link.

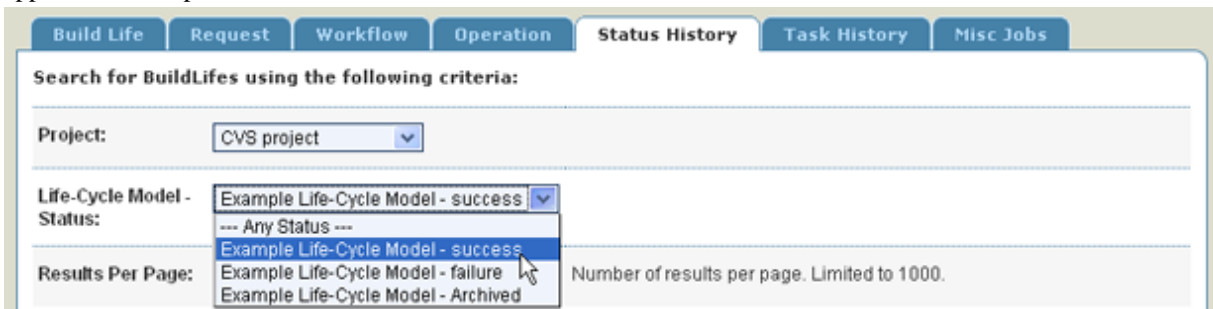


- Click **OK** in the dialogue box.

Lookup Builds by Status

Typically, when promoting a Build Life it is assigned a new status to reflect that change. Reviewing which Build Lives have been promoted, deployed, or released is pretty easy.

- Go to **Search > Status History** tab.
- From the drop-down menu, select the **project** to be searched.
- Select the **Life-Cycle Model status** you are interested in. Only statuses associated with the selected project will appear in the drop-down menu.



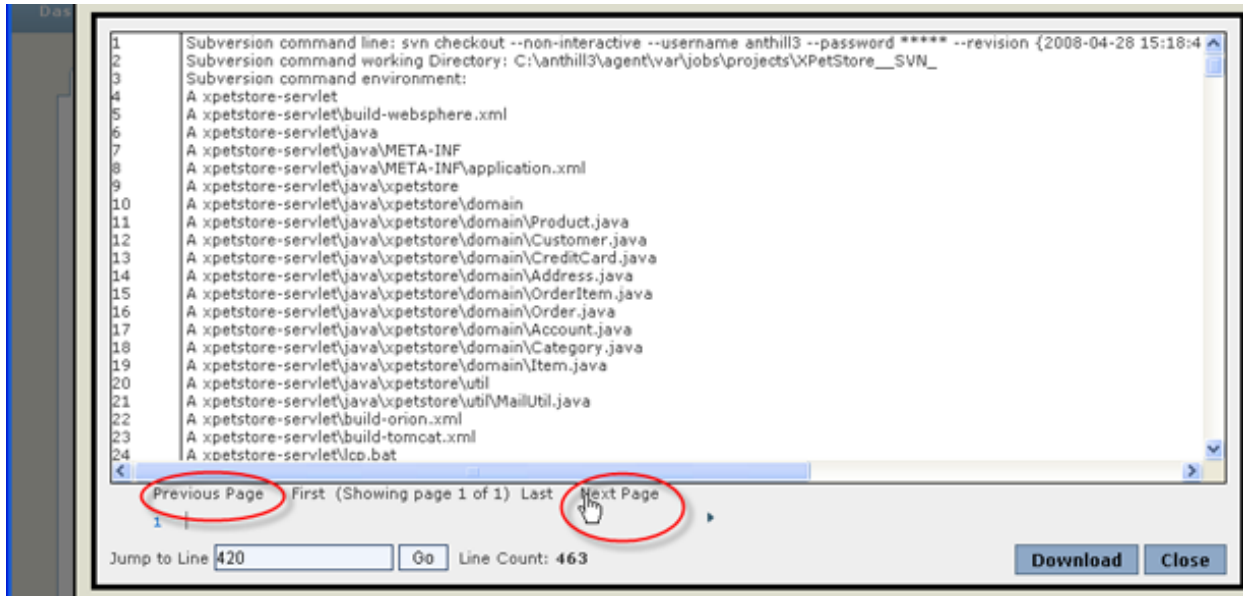
- Click the **Search** button to get a list of **Build Lives**.

5. To view a Project, Build Life, or Job, follow the appropriate link.

Trace a Build Life to Source

In order to be fully traceable and auditable, looking up the sources that a build originated from is important. There are two basic sources of information. The first is to examine the command used to check out the sources. Because checkout commands always uses an exact checkout like a date or baseline, determining what was checked out is straightforward. To do that from the Dashboard, go to the Build Life Main page.

AnthillPro uses an AJAX log viewer, which makes the log available in 1,000-line increments, so you don't have to scroll through the entire log to find what you are looking for. The viewer allows you to jump to a specific line or view the log in 1,000-line increments by clicking the navigation links.



1. Go to **Dashboard** > **Build Life** to be traced.
2. On the **Main Build Life** page, click the **View Job** icon.

Request	Workflow Name	ID	Start Date/Offset	Duration	Env/Selected Agent	Status	Actions
75603	Build Workflow	15137	2007-12-12 03:17:53 PM EST	00:10:05	build farm	Complete	
	Build	35513	00:00:00	00:10:05	cyprus	Success	

Powered by anthillpro (Version 3.4.8-b7)

3. To view the output log of the **Populate Workspace Step**, Click the **Output** icon.

JobTrace Step/Command	Status	Start Offset	Duration	Actions
1. Cleanup Step	Success	00:00:00	00:00:01	
2. Populate Workspace	Success	00:00:01	00:00:01	
populate-workspace-0	Success	00:00:01	00:00:01	
3. Get Changelog	Success	00:00:02	00:00:00	
4. Echo POM	Success	00:00:02	00:00:00	

- The checkout command is listed at the top of the log, and should read something like:

```
Subversion command line: svn checkout --non-interactive --username anthill3
--password *****
http://192.168.1.155/urbancode/xpetstore/trunk .
Subversion command working Directory: C:\anthill4\agent\var\jobs\projects
\XPetStore__SVN_
Subversion command environment:
U    xpetstore-servlet\web\jsp\decorators\default.jsp
Checked out revision 243.
command exit code: 0
```

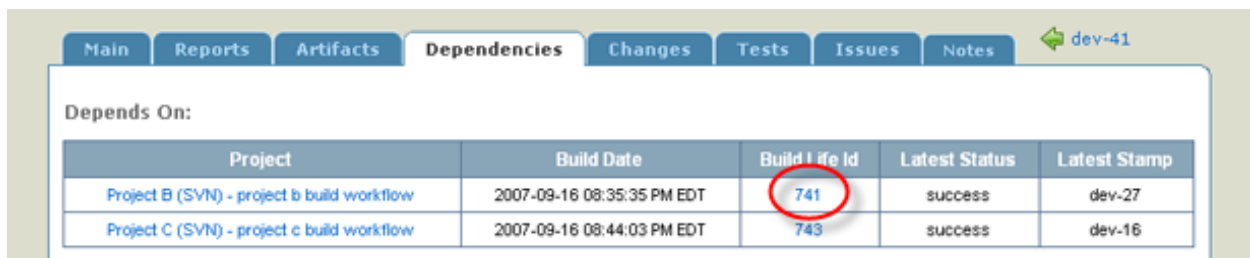
4. Alternately, lookup the label that has been applied to the source code. This is usually a modification of the stamp that was applied, but to be absolutely certain, view the output of the Label Source step. With the baseline label, check out the source tree used and review the source code.

Trace Build Life Dependencies to Source

It's well and good to know the sources used when building the project, but if you can't trace the library another team provided back to its source, there may be a problem. Fortunately, AnthillPro makes it easy to trace dependencies to source.

The Dependency tab on the Build Life page lists the projects your project is dependent on. It also details which Build Life was used, when the dependency was built, as well as the Build Life id and the latest status and stamp attained by the dependency's.

- To view the project dependencies, select the Dependencies tab of the Build Life in question.
- Clicking the dependency's Build Life id, traces its sources the same way sources were looked up in the first section.



Project	Build Date	Build Life Id	Latest Status	Latest Stamp
Project B (SVN) - project b build workflow	2007-09-16 08:35:35 PM EDT	741	success	dev-27
Project C (SVN) - project c build workflow	2007-09-16 08:44:03 PM EDT	743	success	dev-16

Chapter 42. Workflow Tools

There are two major categories of workflow tools: priorities and requests. Workflow priorities allow you to determine the order in which workflows run -- this is especially helpful in busy build farms for top-level projects (e.g., a project used to deploy to production) need to build as quickly as possible. Priorities are set as part of the configuration process.

Workflow requests, on the other hand, are not part of the normal configuration process (see Workflow Requests for more). Rather, they can be controlled using a request plan set up separate from workflow configuration. Once a build has begun, it is possible to see all the requests spawned by a particular build (e.g., requests for builds of dependent projects).

Workflow Priorities

Use workflow priorities to determine which workflow will run first. AnthillPro has three priority levels: High, Normal, and Low. All workflows with a priority of High will run before any other queued workflows with Normal or Low priorities, and workflows set with a Normal priority will run before those with a Low priority. However, once a workflow begins, it will not be suspended or stopped when a higher priority workflow is requested.

It is also possible to set a dynamic workflow-priority based on the Build Life (if applicable), environment, project, request, user, and/or workflow by creating a workflow priority script. See Workflow Priority Scripts.

Workflow Priorities are set during workflow configuration by selecting the priority from the drop-down menu. Additionally, the priority of an existing workflow may be edited on the workflow configuration page. To do so, go to **Administration**, select the workflow, and click the **Edit Workflow** icon on the Main page. Select the new priority from the drop-down and click **Save**. The next time the workflow runs, the new priority will be used.

Workflow Priorities and Dependencies

The workflow priority will cascade down the dependency graph for either pushed or pulled builds. If a workflow with a higher priority is requested as part of a dependency build (see Workflow Priorities with Pulled Builds and Workflow Priorities with Pushed Builds), all subsequent requests will be assigned the highest priority for the remainder of the process. This holds for all dependency-based requests, including running any secondary workflows. However, the hard-coded workflow priority will not be changed. See also Use Workflow Request Contexts.

Workflow Priorities with Pulled Builds

If a request for a workflow within a pulled dependency relationship is made, the highest priority within the workflow context will be assigned to all subsequent workflows requested by the dependency build. For example: Project A depends on Project B; and Project B depends Project C. Project A has a High priority; Project B has a Normal priority; and Project C has a Low priority.

With a request to build Project A -- which has a High priority -- the dependency builds of Projects B and C will be assigned a High priority, inherited from the parent project. Likewise, if a build of Project B is requested (i.e., the build is due to a source change in Project B and not a dependency request), the dependency build of Project C will inherit Project B's Normal priority for the duration of the build. See also Use Workflow Request Contexts.

Workflow Priorities with Pushed Builds

If a request for a workflow within a pushed dependency relationship is made, the highest priority within the workflow context will be assigned to all subsequent workflows requested by the dependency build. For example: Project A depends on Project B; and Project B depends Project C. Project A has a Low priority; Project B has a Normal priority; and Project C has a High priority.

With a request to build Project C -- which has a High priority -- the dependency builds of Projects B and A will be assigned a High priority, inherited from Project C. Likewise, if a build of Project B is requested (i.e., the build is due to a source change in Project B and not a dependency request), the dependency build of Project A will inherit Project B's Normal priority for the duration of the build. See also Use Workflow Request Contexts.

Workflow Properties and Build Life Notes

When performing deployments, AnthillPro automatically tracks who performed the deployment and when the deployment was executed, but not "why" (e.g., a deployment may have been performed to send the artifacts to a staging server, to a specific testing machine, etc.). To track "why" a deployment was performed, use a Text Workflow Property to add an input to the deployment workflow that records why the deployment took place. See Create Workflow Property Note.

In order to make the note easily available as a Build Life Note, also use a simple BeanShell script (that includes the reason) in an Evaluate Script step as part of the deploy job. See Add Script to Workflow Property Note.

When the workflow is run, the property will appear on the Run Secondary Process page, and require the user to input a value. See Run Deployment with Workflow Property Note.

Prerequisites: Creating Workflow Property Notes

- You must have administrative permissions. See Security.
- An active AnthillPro project must have at least one Build Life.
- A deployment workflow must already be created.
- Familiarity with AnthillPro Scripting. See Scripting Basics.

Create Workflow Property Note

To create a workflow property for a Build Life Note:

1. Go to **Administration** and select the **deployment workflow** to add properties to.
2. On the workflow page, select the **Properties** tab.
3. Click **New Property**, select **Text** from the drop-down menu, and click **Select**.
4. Set property:
 - **Name.** The name can be something like "Reason" or "Why". The property <name> will be accessed as `${property:<name>}`. When adding the script, the name given here will be used. See Add Script to Workflow Property Note.
 - **Description.** Enter the question you want to ask. For example, "Why have you decided to deploy this application to this environment?"
 - **Default Value.** This can be left blank when using workflow properties for Notes.
 - **User May Override.** Check the box to allow users to override the property. If checked, the user will be able to specify a new value. If the user may override the property, give the following:
 - **Label.** Give a label for this Property to be shown when prompting users for value (leave blank to use the Name as the Label).

- **Is Required.** Check to require the user to answer the question.
5. Click **Save**.
 6. See Add Script to Workflow Property Note.

Add Script to Workflow Property Note

Add the property to the deployment by using an Evaluate Script step in the deploy job. The script should look up the current Build Life, the current workflow, the user, the environment, and the property. Additionally, the script must also tell AnthillPro to automatically create a Build Life Note. See Build Life Note Scripts.

1. Go to **Administration** and select the **job** associated with the deployment workflow.
2. On the job configuration page, select the **Insert Before** or **Insert After** icon of the step before or after where the script is to be included.
3. Open the **Miscellaneous** folder, select the **Evaluate Script** step, and click **Select**.
4. **Name** the script. For example, "Note".
5. **Description.** Give a description such as "Requires person running a deployment to give a reason."
6. **Script.** Create the BeanShell script that looks up the user requesting the deployment, the environment the deployment is targeting, and then combines that with the reason given by the user. The script also includes a short string that creates the Build Life Note.
 - Note that the property "Reason" is requested by the script. Make sure this matches the name given in Create Workflow Property Note.

See Build Life Note Scripts.

7. Click **Save**.
8. See Run Deployment with Workflow Property Note.

Run Deployment with Workflow Property Note

1. To view the note, the deployment must first be run.
2. Once the deployment has completed, go to the **Dashboard** and select the newly created **Build Life**.
3. On the **Build Life page**, select the **Notes** tab.
4. The note, including the reason given, is displayed.

Workflow Requests

In AnthillPro, the request is the first action taken by the server when executing an originating (i.e., build) or secondary (e.g., deployment) workflow. It does not matter if a person clicks the Build or Run a Secondary Process button or if a schedule or repository trigger kicks off a workflow: All these actions first generate a request before Ant-

hillPro builds a projects or deploys the artifacts. Even if a workflow fails or does not run, a request is generated and recorded.

In AnthillPro, there are two basic tools available for managing/using requests:

- **Request Plan.** Allows you to save the configuration for running one or more originating workflows and then run the build configuration at will. This can come in handy if you have to build many projects that have no dependency relationship but must be delivered as part of the same context. See Use Workflow Request Plans.
- **Request Context.** Available once a request has been completed and the server has begun a build (deployment, etc.), the context lists all the requests spawned by a single workflow. This can come in handy when you need to know which Build Lives were created by a dependency trigger or need to manage the build order. See Use Workflow Request Contexts.

In addition, AnthillPro provides a specialized request search (go to **Search > Request**) for Build Life requests. You can search by a combination of project and workflow requests. The results of a search will tell you if a new Build Life was created or not, or if the request itself failed. You can drill down into each request to investigate requests, etc.

Use Workflow Request Plans

Request Plans are a way to save the configuration for running one or more originating workflows. For example, if you have multiple projects that must be delivered together -- but have no dependency relationships -- you can include all the build workflows into a single request plan. When the request plan is run, it will build (depending on your workflow configurations) your projects. When complete, you can then use the request context to see which projects were build.

There is no restriction on which originating workflows are included in a request plan: the same workflow can be part of multiple request plans and a single request plan can contain workflows from any AnthillPro project. Once configured (go to **Dashboard > Saved Requests**), the request plan can be run whenever you need to. When you create a request plan, it will save the properties that may need to be set when each workflow in the plan runs. The requests created by the request plan will run in the same request context and will take precedence in dependency resolution. To create a request plan:

1. Go to **Dashboard > Saved Requests** and click **Create Request Plan**.
2. Configure the request plan:
 - **Name** the request plan. This name will be displayed on the Saved Request main page.
 - **Description.** Give a short description. You can include the workflow name(s), project name(s), etc.
3. Click **Save**.
4. On the main Saved Requests page, click the Run icon under the Actions menu.

If you need to modify an existing request plan, select the View icon (the magnifying glass) under the Actions menu. You can then delete or add workflows.

Use Workflow Request Contexts

A workflow request context is a collection of requests for workflows that are processed together. If a request triggers the creation of another request due to dependencies, the requests will be in the same context. For example, all requests that are created by the occurrence of a schedule are created in the same context; and any requests created by a

Run Another Workflow or Run Dependency Workflow step are created in the same context. The request context is most helpful if dependencies are configured in AnthillPro (see Configuring Dependencies): the Request Context shows all the requests for a build spawned by a single action.

By viewing the request context, administrators can easily determine how AnthillPro manages build order and consistency along a dependency hierarchy by using the request context page. The request context page displays the project, workflow, environment, status, priority, and date of each request within the context.

- The request being investigated will be highlighted with an orange frame.
- To view the request for any workflow within the context, click the View Request icon.
- To view the Build Life that was created by the request, follow the link at the Status item.

Request 461

Project: C
Workflow: C Trunk Build Workflow
Environment: C
Status: Created New Build Life -> **Complete**
Priority: Normal
Date: 5/8/09 11:38 AM

CREATED

Request 462

Project: D
Workflow: D Trunk Build Workflow
Environment: D
Status: Created New Build Life -> **Complete**
Priority: Normal
Date: 5/8/09 11:38 AM

CREATED

Request 464

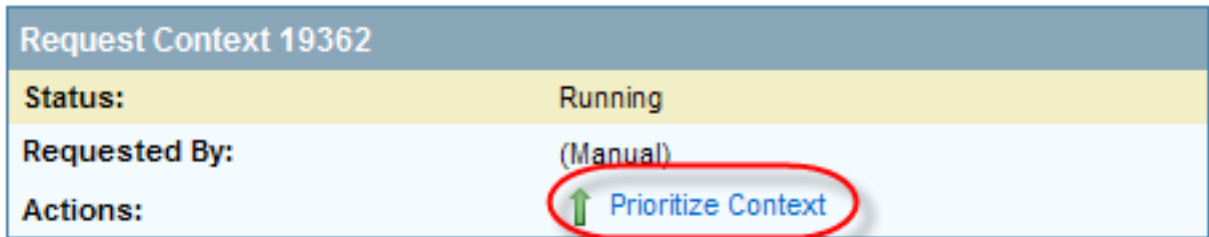
Project: C
Workflow: C Verify Build Workflow
Environment: Build-Farm
Status: Started Workflow -> **Complete**
Priority: Low
Date: 5/8/09 11:39 AM

CREATED

Request 463

Project: D
Workflow: D Verify Build Workflow
Environment: Build-Farm
Status: Started Workflow -> **Complete**
Priority: Low
Date: 5/8/09 11:38 AM

- To prioritize every request within a running context, click the Prioritize Context link in the Request Context information box. This will automatically assign a High priority to every request within the context.

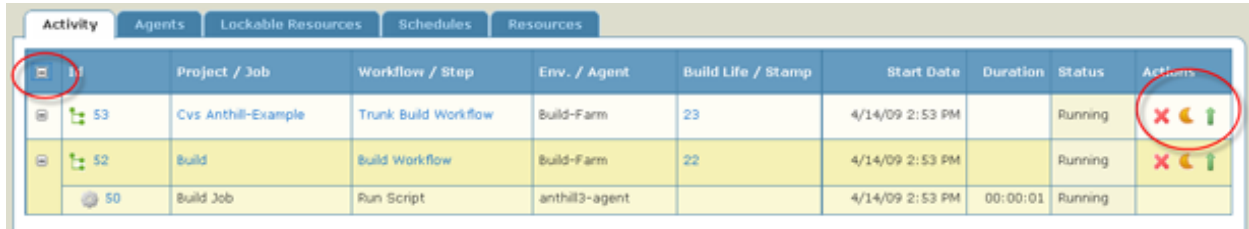


To access the workflow request context:

1. Go to the **Dashboard** and select the appropriate **Build Life**.
2. Click the **Request Number** under the Request menu.
3. On the Request page, click the **View All Requests in Context** icon in the Build Request menu.

Chapter 43. Job Tools

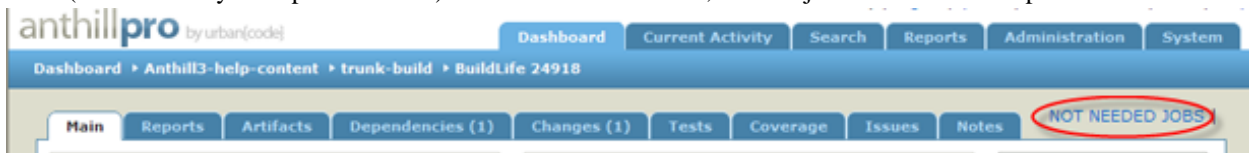
The **Activity** tab shows all the current actions the AnthillPro server is performing, as well as provides an opportunity to abort, suspend, or prioritize jobs under the **Actions** menu. Following the **ID**, **Project**, **Workflow**, or **Build Life/Stamp**.



	Project / Job	Workflow / Step	Env. / Agent	Build Life / Stamp	Start Date	Duration	Status	Actions	
+	53	Cvs Anthill-Example	Trunk Build Workflow	Build-Farm	23	4/14/09 2:53 PM	Running	X C I	
-	52	Build	Build Workflow	Build-Farm	22	4/14/09 2:53 PM	Running	X C I	
	50	Build Job	Run Script	anthill3-agent		4/14/09 2:53 PM	00:00:01	Running	

Clicking the plus (+) sign in the upper left hand corner will expand/collapse all workflows, or an individual workflow may be expanded/collapsed. Additionally, current activity may be aborted, prioritized or suspended by using the icons under the Action menu.

Selecting the **NOT NEEDED JOBS** link for any Build Life allows you to view jobs that did not run as part of the build (determined by their pre-condition). If the link is not visible, then all jobs were needed as part of the workflow.



Chapter 44. User Views

Each AnthillPro user has some control over what is displayed on the Dashboard; however, what is available is determined by the AnthillPro administrator. In the User Profile, you can add/change your password; user-repository alias; first and last name; e-mail and IM addresses; time zone; and the number of dashboard rows displayed. Any changes made in the Profile will be updated in the AnthillPro security system. See Change User Profile.

Additionally, if the system administrator has elected to have the Dashboard Graphs visible (see Configure Server-wide User Views), you can disable them by selecting the Views tab. See Change User Views.

Change User Profile

On the User Profile General tab, update contact information, update password, and/or set up a user alias. *Note that the Name filed (denoted by a red asterisk) cannot be changed here.* If you need to change the name, contact your AnthillPro administrator.

To change your profile settings:

1. Select the **profile** link from the menu at the top of the browser window.



2. **Update Password.** On the General tab, select the Update Password button to change your AnthillPro password. In the pop-up window, give:

- Your **current password**.
- The **new password** you want to use when accessing AnthillPro.
- A **confirmation** of your new password.
- Click **Update**.

Once reset, the new password will be required at your next login.

3. To change name and other user information, click the **Edit** button. You can then edit your contact information as well as the date and time formats. Note that any changes made here will be reflected in the AnthillPro security system.
4. **Number of Dashboard Rows.** To change the number of builds displayed on the Dashboard Workflow page, input it here. For example, if you input 5, only the five most recent builds will appear under the Recent Activity menu on the Main tab.
5. **Add User-Repository Alias.** To add a new **alias**, click the **User-Repository Alias** button. Select the appropriate **repository** from the drop-down menu and give the alias used in that repository. Click the **Add User-Repository Alias** button to finish.

Once created, a User-Repository Alias may not be edited. If your user alias for a repository has changed, delete the current alias (click the **Remove** icon under the Operations menu) and create a new one.

6. Click **Save**. If not changing Views, click **Done**. Otherwise, see Change User Views.

Change User Views

If your system administrator has elected to have the Dashboard Graphs visible (see Configure Server-wide User Views), you can disable them on the Views tab. However, if the administrator has already disabled the graphs, you will not be able to display them.

To enable/disable Dashboard Graphs:

1. Select the **profile** link from the menu at the top of the browser window.



2. Select the **Views** tab and click **Edit**.

3. Configure settings:

- **Main Dashboard Graphs.** Select **Yes** to have the Total Source Activity, Build Activity, and Average Test Success graphs displayed on the Main page; or **No** to disable them.
- **Project Dashboard Graphs.** Select **Yes** to have the Source Activity, Build Activity, and Average Test Success graphs displayed on the Project page; or **No** to disable them.
- **Workflow Dashboard Graphs.** Select **Yes** to have the Source Activity, Build Activity, and Average Test Success graphs displayed on the Workflow page; or **No** to disable them.

4. Click **Save**. If not changing General settings, click **Done**. Otherwise, see Change User Views.

Use Enable Refresh

To get the most up-to-date information about your projects, use **ENABLE REFRESH**. When activated, ENABLE REFRESH will reload the Dashboard every 30 seconds. Any change in project, workflow, or Build Life status will be updated automatically.

1. Go to the **Dashboard**.
2. Select the **ENABLE REFRESH** link.



3. Once enabled, you can either **DISABLE** or **PAUSE** refresh by selecting the appropriate link.

If refresh has been paused, it may be restarted by selecting the **RESUME** link. If refresh has been disabled, it may be restarted by selecting the **ENABLE REFRESH** link again.

Refresh in 20 seconds...

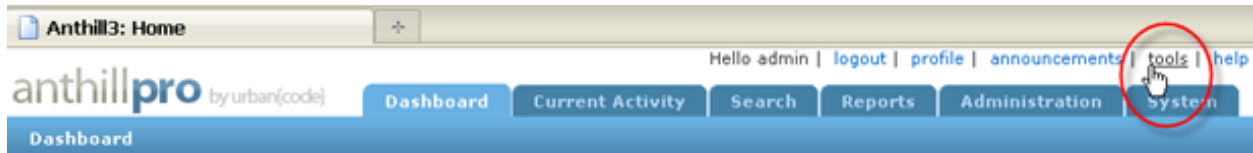
[DISABLE REFRESH](#) | [PAUSE](#) | [RESUME](#)

89% successful	Success Rate: 308/344
	Avg. Duration: 0 s

Chapter 45. System Tray Monitor

The System Tray Monitor provides feedback directly to the desktop, without having to open or refresh a browser. If there is a problem with a selected action, the Monitor will display an error telling you to check the configurations for that task. Once configured, the Monitor will save your settings, so if you exit and restart, it will continue to monitor your selections.

The System Tray Monitor, including complete documentation, is available from the AnthillPro UI at **Tools > System Tray Monitor**.



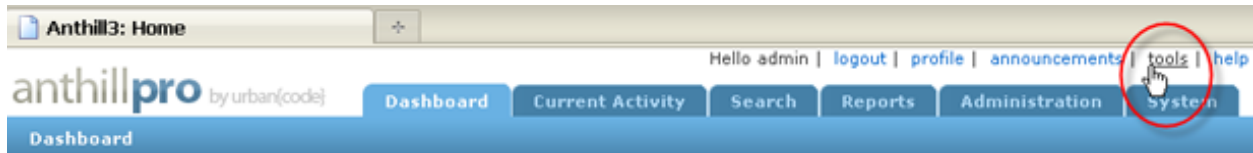
Part IX. Developer Tools

AnthillPro provides a number of user and developer tools in addition to the UI. The tools allow any user to install a system tray monitor and provide developers -- via the Dev-kit -- access to the API and Codestation. In addition, there are Visual Studio and Eclipse plugins and preflight builds for developer productivity.

Current AnthillPro tools:

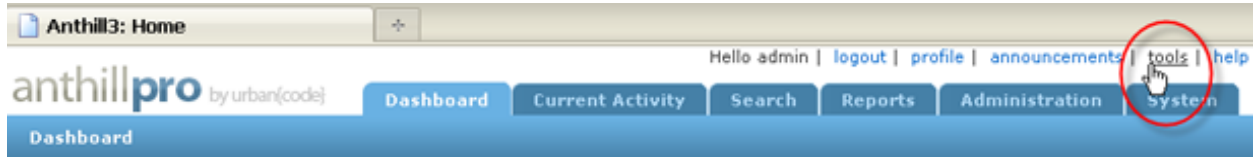
- **Dev-kit.** The Dev-kit provides comprehensive documentation on AnthillPro scripting, remoting, using SOAP-based web services with AnthillPro, the AHPTool (command-line tool) and creating your own Plugin for AnthillPro.
- **IDE Plugins.** AnthillPro provides IDE Plugins for both Eclipse and Visual Studio. The Plugins enables developers to view the current activity and state of projects, start new builds, map their projects to projects in AnthillPro, and resolve the project's dependency artifacts.
- **Preflight Builds.** Allows developers to run a test build in the authoritative build environment before committing their changes to source control.
- **Codestation (Developers).** Provides documentation for advanced usage of Codestation, including Ant tasks, the command-line interface, and Codestation properties.

Complete documentation and tools are available by following the **tools** link:



Chapter 46. Dev-kit

The Dev-kit provides comprehensive documentation on AnthillPro scripting, including the full API and the AnthillPro script library; remoting, which provides full access to the AnthillPro object model; using SOAP-based web services with AnthillPro; as well as instruction on creating your own Plugin for AnthillPro.



To view the Dev-kit, follow the tools link in the upper right corner of the UI and download the **Development Kit Bundle**.

Scripting

The scripts in AnthillPro are written in BeanShell, which is a JSR-approved Java scripting language. Other scripting languages are also supported for select activities. The Scripting section of the Developer Tools provides a basic introduction to scripting and AnthillPro.

Urbanocode also maintains a public Script Database [https://bugs.urbanocode.com/secure/IssueNavigator.jspa?reset=true&pid=10110&status=5] that includes sample scripts for custom reports, steps, reusable snippets, and more.

Detailed instruction, available at **Tools > Development Kit Bundle > scripting**, includes documentation on the AnthillPro:

- **Script Library.** Use the script library to create, organize, and provide security around often-used AnthillPro scripts. The Script Library is most helpful for large organizations, allowing them to ensure that only the appropriate team members can modify a script.

Remoting

Remote scripting provides full access to the AnthillPro object model via a remote interface. This provides access to all of the administration features of AnthillPro, as well as the information that is available via the Dashboard. Of course, security (authentication and authorization) are fully enforced when using the remote scripting API.

Detailed documentation is available at **Tools > Development Kit Bundle > remoting**.

Integrating Using SOAP

Anthill Pro provides a SOAP-based, web services facade to enable third-party and user-created applications to cleanly integrate. The SOAP interface, available at launch, will provide a number of lookup utilities to provide information on system state, project progress, and current activity. Also, as standard build and deploy web services vocabularies evolve in the Eclipse Application Life-Cycle Framework project, this SOAP interface will provide whatever functionality is specified.

Detailed documentation is available at **Tools > Development Kit Bundle > soap**.

Developing Plugins

You can write your own integration with third-party tools (such as testing, SCM, source-code analytic, etc., tools) and then add them to your AnthillPro workflows.

For an introduction to the Plugin system, see also [Using AnthillPro Plugins](#).

Detailed documentation is available at [Tools > Development Kit Bundle > plugin](#).

AHPTool

AHPTool is a command-line interface for AnthillPro that provides communication between agent-side commands, scripting and the AnthillPro central server. AHPTool is focused on setting and retrieving information from the AnthillPro server in the context of a running workflow: It can be used to look up or set properties at the system, step, request, job, Build Life and agent levels. AHPTool can upload and retrieve Test, Coverage, Analytics, or Issue data in the form of an XML document to AnthillPro -- making it an excellent integration point for writing your own Plugin or script.

Detailed documentation is available at [Tools > Development Kit Bundle > ahptool](#).

AnthillPro API

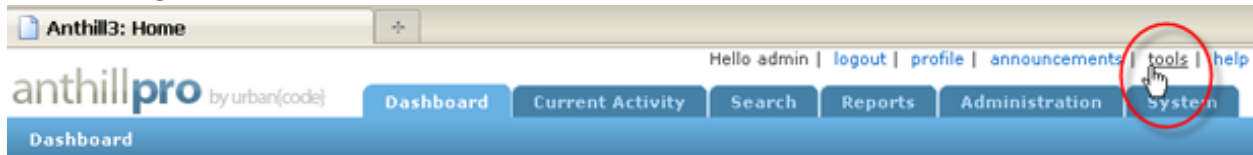
The API is available under the server installation through the web UI. The majority of the API is available at the tools page, including a copy of the AnthillPro API documentation (JavaDoc). The objects in the API strongly reflect what is seen in the user interface.

In addition, AnthillPro also has a REST API that allows you to access AnthillPro's web services as resources, via their URLs, using standard HTTP protocol. The responses to most of the GET methods (except property values, which are simple text) are in XML, and may be consumed by any application, etc., that is able to consume XML files.

Detailed documentation is available at the [Tools](#) page.

Chapter 47. IDE Plugins

AnthillPro provides IDE Plugins for both Eclipse and Visual Studio. The Plugins enables developers to view the current activity and state of projects, start new builds, map their projects to projects in AnthillPro, and resolve the project's dependency artifacts. The Plugins are available from the AnthillPro UI at **Tools > Eclipse Plugins** or **Visual Studio Plugins**.

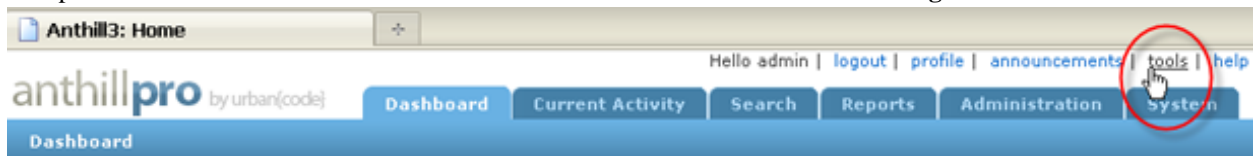


Chapter 48. Preflight Builds

Preflight builds are created on the Administration tab by selecting the Create Preflight Workflow icon of the appropriate originating workflow. Once created, the preflight-build workflow is displayed on the Administration page in blue and identified with the (preflight) suffix. Both the preflight workflow and job may be edited after creation; however, making changes to either the preflight workflow or job may result in a failed preflight-build.

While the preflight workflow and job are virtually identical to the originals, the preflight job includes an additional step (Preflight Build - File Retrieval) which is inserted after the first populate workspace step encountered in the original job. When the job is run, the original populate workspace steps are disabled (but not removed), and the Preflight Build -File Retrieval job step instructs AnthillPro to retrieve source files for a developer machine, instead of using the SCM configured for the originating workflow. Any additional populate workspace step can be enabled for the preflight build, if needed; however it should not interfere with the one actually replaced. Any assign stamp, label, change log and integration step is disabled permanently for preflight jobs.

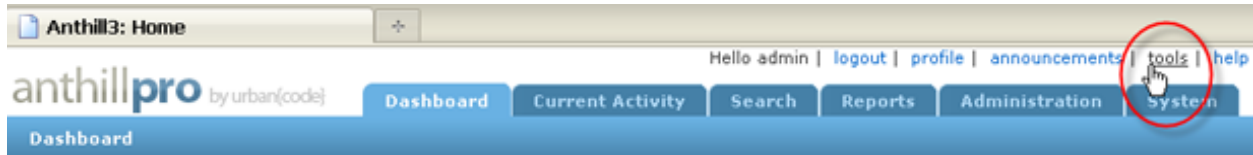
Complete documentation is available from the AnthillPro UI at **Tools > Anthill Preflight Client**.



Chapter 49. Codestation for Developers

Codestation is the name of AnthillPro's artifact repository management and access tool set. It provides support for dependencies of third-party tool kits and software libraries. Codestation is also responsible for bringing the dependency management lookup and retrieval utilities to the individual developer.

AnthillPro provides a number of Ant tasks and a command-line interface for developers. The tools, and full documentation, are available by following the **tools** link:



There are also a number of advanced settings and features, outlined below, to change Codestation's behaviour. For most users, the default settings should suffice.

Codestation, Artifact Sets, and Exclusive Locks

When uploading artifacts to a Build Life, Codestation exclusively locks the Build Life for the duration of the process. Once the upload begins, all other requests (from other workflows, etc.) to upload artifacts to the locked Build Life must wait for the lock to release before the subsequent upload begins.

Under certain conditions Codestation may throw the following error: `Server Error: HTTP Code 500: Could not acquire exclusive access to the buildlife.` If you see this error when Codestation is attempting to upload multiple artifacts to the same Build Life, the first upload most likely exceeds the 15-second default time-out. To address this issue, increase the time-out setting. For example, if the longest upload takes 30 seconds, the time-out setting must be greater than 30 seconds, otherwise Codestation will throw an error. To reset the Codestation time-out:

1. Shut down the AnthillPro server.
2. Open the server's **bin/ah3server** (UNIX) or **bin/ah3server.cmd** (Windows) file in a text editor.
3. Add the following to **JAVA_OPTS** (under BEGIN INSTALL MODIFICATIONS):

```
-DCodestationRepositoryFileHelperServer.tryLockTimeoutSeconds=
```

After the equals sign (=), give the time, in seconds, that Codestation should wait before timing out. For example, input **=60** to have Codestation wait one minute before timing out.

4. If AnthillPro is running as a Windows service, repeat Item 3 in the **bin/service/ah3server.cmd** file.

Once modified, the Server must be re-installed as a Windows Service. See [Manually Install the AnthillPro Server as Windows Service](#).

5. Restart the AnthillPro server.

Codestation Artifact Time-to-live

Use the **Codestation Time-to-live** feature to determine how long unused artifacts remain in the cache. By default,

every time the cache is accessed, artifacts that have been unused for 3 days will be automatically removed.

The Time-to-live option can be configured either in the **codestation.properties** file, or as an **agent property**. For both configurations, the value equals the number of days unused artifacts are to be stored in the cache. For example, in the `codestation.properties` file, `codestation.cache.timeToLive=10` will delete unused artifacts after 10 days.

For the `codestation.properties` file:

1. Open the **codestation.properties** file (located in the user home directory).
2. Configure with the property `codestation.cache.timeToLive`.

If the property is not set (or is set to zero), unused artifacts will be removed after 3 days. To never delete the artifacts, set the property to a negative number.

Example properties file:

```
codestation.server=https://anthillserver.company.com
codestation.user=build
codestation.password=buildpass
codestation.noCheckCertificate=false
```

For the Agent (you must have administrative permissions to the agent):

1. Go to **AnthillPro UI > Agents > Agent** and select the appropriate agent from the list.
2. Select the **Agent Properties** tab and create a new property.
3. Give the property name and value in the **name=value** format (e.g., `codestation.cache.timeToLive=10`).

If the property is not set (or is set to zero), unused artifacts will be removed after 3 days. To never delete the artifacts, set the property to a negative number.

4. **Name.** Same name as the property (`codestation.cache.timeToLive`).
5. **Value.** The path of the cache directory.

Codestation BOM Suppression

Normally a Codestation resolve creates BOM files that describe what artifacts were part of the resolve so that they can be cleaned up for the next resolve. In scenarios where the cleanup is never performed, the BOM files are unnecessary. If using the BOM suppression feature (e.g., to clean up previous deployments, etc.), the agent must be set to look for the `codestation.suppress.boms` variable.

The Codestation BOM suppression feature can be configured either in the **codestation.properties** file, as an **Ant task**, or on the **command-line tool**:

- **codestation.properties file.** Open the **codestation.properties** file (located in the user home directory) and configure property.

Example properties file:

```
codestation.server=https://anthillserver.company.com
codestation.user=build
codestation.password=buildpass
codestation.noCheckCertificate=false
```

- **Ant task.** Configure on Ant tasks with a boolean `suppressBoms` property. See **Tools > Codestation Client**.
- **Command-line tool.** Configured on the command line tool with the `suppressBoms` option. See **Tools > Codestation Client**.

Set the agent to look for a boolean variable called `codestation.suppress.boms`. Once this has been set, the agent will ignore the **codestation.properties** file.

1. Go to **AnthillPro UI > Agents > Agent** and select the appropriate agent from the list. You must have Administrative permissions to add an agent property. See **Manage Security**.
2. Select the **Agent Properties** tab and create a new property.
3. Give the property name, which is the same name as the property: `codestation.suppress.boms`.

Codestation Fallback to Offline

Use the Fallback-to-offline option to automatically switch Codestation offline if an error occurs while contacting the AnthillPro server. With this option, Codestation will only switch offline for the duration of a given Codestation command. Once the command completes, future commands will attempt to communicate with the server as normal.

The Fallback-to-offline option can be configured either in the **codestation.properties** file, as an **Ant task**, or on the **command-line tool**. Set the value for this option as "true" or "false".

- **codestation.properties file.** Open the **codestation.properties** file (located in the user home directory). Configure with the property `codestation.fallbackToOffline`.

Example properties file:

```
codestation.server=https://anthillserver.company.com
codestation.user=build
codestation.password=buildpass
codestation.noCheckCertificate=false
```

- **Ant task.** Configure on Ant tasks with the `fallbackToOffline` attribute. See **Tools > Codestation Client**.
- **Command-line tool.** Configure on the command line tool with the `-fallbackToOffline` option. See **Tools > Codestation Client**.

Codestation Secondary Cache

If remote teams have slow connection speeds, add the Codestation Secondary Cache on top of the currently supported local cache to improve performance. Once the secondary cache is set up on a network share (on the LAN), artifacts are pulled from AnthillPro once, and then shared among all the developers at the remote location. (Without the Secondary Cache, each developer must pull his/her own copy.)

For the `codestation.properties` file:

1. Open the `codestation.properties` file (located in the user home directory).
2. Configure with the property `codestation.secondaryCache.dir`.

Example of properties file:

```
codestation.server=https://anthillserver.company.com
codestation.user=build
codestation.password=buildpass
codestation.noCheckCertificate=false
```

For the Agent (you must have administrative permissions to the agent):

1. Go to **AnthillPro UI > Agents > Agent** and select the appropriate agent from the list.
2. Select the **Agent Properties** tab and create a new property.
3. Give the property name and value in the **name=value** format.
4. **Name.** Same name as the property (`codestation.secondaryCache.dir`).
5. **Value.** The path of the cache directory.

Codestation File-defined Dependency Configurations

You can use a Codestation method to set the dependencies of a Build Life from a file-defined resolve and then add it to the client and CLI. File-defined dependencies, using an XML file, is supported for both historical resolves and transitive dependencies. Once your file is configured, you can have Codestation pull down the dependencies.

For more, go to **Tools > Developer Tools > Codestation Client**.

Part X. Server Management

Once the server is started (by running **start_ah3server.cmd** in the \bin directory), configure there are a number of system-level network settings, the security settings, miscellaneous settings, and user views you can configure.

The installation process also created default values that enable the server and agents to communicate. However, additional network configuration may be required depending on your existing systems, or if the values (especially for server-agent communication) conflict with your existing networks. When changing settings, it may be necessary to restart the server for the new configurations to take effect, as well as changing agent settings (especially with SSL).

In addition, it is possible to view the server log on the Log tab. Depending on the logging level, the server log may be used for debugging, gathering general server information, etc.

Chapter 50. Server Settings

Before you begin changing any server settings, make sure you have full access to the **System > Server** menu. If you are changing any communication settings, make sure you already have an agent installed, configured, and online -- this will make troubleshooting any problems easier if an agent has already been able to successfully connect to the server. Server settings are broken-up into a few categories as follows:

- **Network.** The system-level network settings used to configure communication between the AnthillPro server and agents. See Server Network Settings.
- **Security.** The system-level security settings used to configure access to the Anthill server settings for configuration and artifact management. See Server Security.
- **Misc.** The system-level miscellaneous settings include options for repository cache, dependency conflicts, and server logging. See Server Miscellaneous Settings and Server Logging Options.
- **Views.** Disable or enable the views for all AnthillPro users. See Server-wide User Views.

In addition, you can set system (formerly server) properties. See System Properties.

Server Network Settings

Configure the system-level network settings used to configure communication between the AnthillPro server and agents.

1. Go to **System > Server Settings** under the Server menu and click **Edit**.
2. Select the **Network** tab and click **Edit**.
3. Configure settings:
 - **Bind to IP.** During the installation process, the **External IP** address where the **agent** can be reached was set. If no specific IP address was specified, then AnthillPro will use 0.0.0.0 and bind to all available addresses (this is recommended).

If a specific IP was designated during agent installation, select it. Server-agent communication will then only occur on the bound IP, and any agent with a different IP will not communicate with the server.

Restart the server if a new IP is chosen.
 - **Server IP.** Give the IP address used to connect to the AnthillPro server for remoting (if using remote scripts) and/or the Distributed web (if using Distributed Servers).
 - **External URL.** Enter the URL used to access AnthillPro (e.g., `http://anthillserver/`). The external URL is used for notifications and to access AnthillPro from remote sites, and may include `http(s)://` or any non-standard ports.
 - **Agent External URL.** Enter the URL agents use to access the web application to return log files and upload reports to the server. This includes `http(s)://` and any non-standard ports (e.g., `http://myserver/` or `http://myserver:8080/`). The same external URL may be used for both agents and other external access. If AnthillPro is running behind a web server or a firewall, using a different external URL for the agents will provide greater security, etc.
 - **Keystore Password.** Reset the keystore password.

- **Use SSL Between Server and Agents.** See Server-Agent SSL before changing this option.

Restart the server if SSL is changed.

- **Enforce Mutual Authentication.** See Server-Agent SSL before changing this option.

Restart the server if Mutual Authentication is changed.

4. Click **Save**. Click **Done** if not configuring other server settings.

Server Security

Set the system-level security settings used to determine access to the AnthillPro server settings for configuration and artifact management.

1. Go to **System > Server Settings** under the Server menu.

2. Select the **Security** tab and click **Edit**.

3. Configure settings:

- **Digest Algorithm.** Choose a digest algorithm from the drop-down menu. AnthillPro uses either an SHA or MD cryptographic hash function to protect the build and deployment artifacts. If none is chosen, no digest algorithm will be used.

If you are using the **Transfer Only Changed Files** option for your Resolve Artifact step, a digest algorithm must be selected. AnthillPro will use the generated file checksums for the resolve, and skip the Codestation caching on the agent. See also Create a Deployment Job.

- **Audit User Transactions.** Check the box to log all user transactions and changes in the database. When enabled, the log is accessible by going to **System > Audit**. See Perform Audits.
- **Allow Multiple Sessions Per User.** Check the box to allow any AnthillPro user to be concurrently logged in from multiple computers and/or browsers. If left blank, when a user logs on to AnthillPro using a different browser or machine, the current session will automatically expire.
- **Allow Use of Login Cookie.** Check the box allow users to login via a cookie based on prior login with the same computer and browser.
- **Allow Auto-Complete Feature in Web Browsers.** Check the box to allow web browsers to use their auto-complete feature in forms. Auto complete is disabled where higher security is considered.
- **Allow Anonymous Guest Access.** See Set Up and Manage Guest Users.
- **Show Error Trace in UI.** Check the box to show server errors in the UI. To hide error traces in the UI, leave this field blank.
- **Secure Artifact Sets.** By default, any user with access to a workflow can download/resolve the artifacts. If you want to change this functionality, please see Securing Artifact Sets and Configuring Default Permissions if you have not already done so.

Before you change this setting, ensure the Default Permissions are correctly setup for artifact sets. Once you enable this setting, every existing artifact set will inherit the Default Permissions. In addition, each newly created artifact set will also inherit the default permissions.

See also Artifact Set Security for information on securing individual artifact sets at the Life-Cycle-Model level.

4. Click **Save**.
5. Click **Done** if not configuring other server settings. To configure other settings, see Server Settings.

Server Miscellaneous Settings

Set system-level miscellaneous settings include options for repository cache, dependencies, server logging, etc.

1. Go to **System > Server Settings** under the Server menu.
2. Select the **Misc** tab and click **Edit**.
3. Configure settings:

- **Repository Event Cache Time.** Set the length of time in minutes between the oldest and latest repository events that AnthillPro caches. The set-time must be greater than any quiet period *plus* the longest time required to obtain a revision log.

For example, if the longest quiet period for any project is set to 5 minutes and the longest time it takes to obtain the necessary revision logs for any project is 3 minutes, the Repository Event Cache Time must be greater than 8 minutes (i.e., at least 9 minutes). Otherwise, AnthillPro will not accurately detect changes when building dependent projects.

The time it takes AnthillPro to get the revision logs can vary significantly, depending on server load and the complexity of the dependency relationships. *Unless experiencing performance issues because the Repository Event Cache Time is too long, it is advisable to use the default.*

- **Preserve Artifact Time Stamps.** Check the box to preserve file time stamps when transferring artifacts between the server and agents when using the resolve and deliver artifact steps.
- **Include Artifact Directories and Symlinks.** Select this option to include empty directories and symbolic links (and soft links for Unix) when publishing or resolving artifacts. This allows you to publish a link, and not the actual artifacts if desired. *Symlinks can only be published or resolved on Unix agents.*

Including a symlink in your working directory that points to your root directory may result in the deletion of your root directory when AnthillPro's cleanup process executes.

To use this feature, add the link/directory as part of your include pattern. See Capture and Deliver Build Artifacts and Edit the Dependency Workflow.

- **Include Permissions.** Select this option to include permissions when publishing or resolving artifacts. *Note that permissions can only be published or resolved on Unix agents.*
- **Dependency Conflicts.** Select either to detect all dependency conflicts even if the dependencies remain unresolved or to only detect resolved conflicts.

DETECT ALL allows AnthillPro to detect all dependency conflicts that are not resolved by AnthillPro, in addition to those AnthillPro resolves. For example, if using a separate tool to resolve dependencies, this option allows you to view those dependencies.

DETECT RESOLVED only detects dependency conflicts that are resolved by AnthillPro.

- **Cascade Properties to Dependencies.** By default, AnthillPro does not cascade properties to any requests made because of a dependency relationship. If you need AnthillPro to cascade properties from a request to a created dependency request, check the box. This only applies to requests created from pull and push dependencies. This sets the default value, and may be overridden when configuring dependencies.
- **Explicit Dependency Artifact Resolve and Verify.** Select here if you want AnthillPro to download and verify dependencies in separate commands with a command for each artifact set download and each verify. The default is to not check this and do the download and verify in one command on the agent. Checking this can cause dependency resolution to take longer.
- **Default Dependency Type.** Select the default dependency relationship for newly configured dependencies. The user creating the dependency will be able to change the dependency type, but if they forget, this will be the default setting.
- **Default Quiet-period Type.** Select the default project quiet period type for newly configured projects. The user creating the project will be able to change the quiet period type, but if they forget, this will be the default setting. *Before changing the defaults, see Use Agent Filters and Quiet Periods and Workflow Triggers and Scheduled Builds if you are unfamiliar with quiet periods and triggers.* You can choose:
 - Changelog.
 - Repository.
 - None.
- **Logging Level.** Check the logging level for the server. The log is available at **System > Server Settings > Log** and **System > Server Settings > Error**. See Server Logging Options.

ALL	Logs all server events. Because this option produces a large log file, it is <i>recommend for use only when tracing errors</i> .
DEBUG	Used temporarily for debugging server events. Because this option produces a large log file, it is <i>recommend for use only when tracing errors</i> .
INFO	Provides basic information regarding server activity. <i>Recommended for day-to-day use</i> .
WARN	Logs any warnings that are thrown by the server. <i>Recommended for day-to-day use</i> .
ERROR	Logs any errors that are thrown by the server.
FATAL	Logs all fatal errors that may occur.
OFF	Never logs server events.

4. Click **Save**.

5. Click **Done** if not configuring other server settings. To configure other settings, see Server Settings and Server Logging Options.

Server-wide User Views

Disable or enable the views for all AnthillPro users. By default, the Source Activity, Build Activity, and Average Test Success graphs are enabled. You may elect to disable all graphs (on the Main, Project, and Workflow pages), or a combination of the three. Once disabled, no AnthillPro user will be able to view the graph.

If the graphs are enabled, individual users may elect to turn them off (see Configure User Profile); however, once a graph has been disabled at the System level, no AnthillPro user will be able to view the graph until it is enabled here.

1. Go to **System > Server Settings** under the Server menu.
2. Select the **Views** tab and click **Edit**.
3. Configure settings:
 - **Main Dashboard Graphs.** Select **Yes** to have the Total Source Activity, Build Activity, and Average Test Success graphs displayed on the Main page; or **No** to disable them.
 - **Project Dashboard Graphs.** Select **Yes** to have the Source Activity, Build Activity, and Average Test Success graphs displayed on the Project page; or **No** to disable them.
 - **Workflow Dashboard Graphs.** Select **Yes** to have the Source Activity, Build Activity, and Average Test Success graphs displayed on the Workflow page; or **No** to disable them.
4. Click **Save**.
5. Click **Done** if not configuring other server settings. To configure other settings, see Server Settings.

Chapter 51. Backups of AnthillPro

Regular backups of AnthillPro is advisable to protect your data. While AnthillPro supports automated backups with the Derby database (see Backups with Derby), other databases must be backed up using the backup tool that came with the particular database (see Backups with Other Databases).

The var\db directory should never be backed up with a file system backup (it is the location of the database files) because this will almost always result in file corruption.

- Because the Agent does not store critical data (other than agent configurations), Agent upgrade is unnecessary.

Backups with Derby

AnthillPro supports scheduled backups of the Derby database through the UI, using a backup schedule. The backup interval will vary depending on the size of your organization, number of artifacts, etc. Backup schedules may be deactivated or changed on the Server Settings page. See Deactivate or Change Backup Schedule.

The var\db directory should never be backed up with a file system backup (it is the location of the Derby database files) because this will almost always result in file corruption.

Backups are saved in the var\db\backup directory (e.g., C:\AnthillPro\server\var\db\backup\). Every update will be named according to the date and time of the backup (in the format year-month-day_hour-minute-second; i.e., similar to 2008-10-08_10-59-48).

To restore AnthillPro to a clean state from backup files, overlay the installation with the backup files. See Restore AnthillPro to a Clean State.

- Because the Agent does not store critical data (other than agent configurations), Agent upgrade is unnecessary.

Backup Prerequisites (Derby)

- You must have read and write permissions to the System page. See Manage Security.
- Before editing a backup setting, create a special schedule or use an existing schedule. See Create a Backup Schedule.
- *The var\db directory should never be backed up with a file system backup (it is the location of the Derby database files) because this will almost always result in file corruption.*

Create a Backup Schedule (Derby)

1. Go to **System > Schedules** under the **Project Support** menu.
2. Click the **Create Schedule** button to create a new schedule.
3. For backups, creating a basic schedule with a rather long interval works well. Select the **Interval Schedule** option and click **Set**.

Choose Schedule Type

Configure the Schedule Type in this section. Once a Schedule Type is selected, additional properties specific to that Schedule Type will be displayed.

- **Interval:** A schedule that executes at a regular interval during the course of a day.
- **Cron Expression:** A schedule that executes at an interval defined in a cron expression.

Cron Expression Schedule A schedule that fires based on a cron expression

Interval Schedule A schedule that fires after a fixed interval

4. Configure the Schedule.

- **Name.** the schedule.
- Provide a **description** (optional).
- **Set the interval in minutes.** An interval of 1,440 minutes will trigger a backup once a day.
- **Give the start time.** Set the backup to start while system loads are low. A start time of 23:30 will trigger a backup to start at 11:30 p.m.

Setting a start time is most useful for infrequent schedules, such as one that triggers twice a day, where the first daily execution dictates the second.

The Interval Schedule Type allows you to configure schedules based on a given time interval. *denotes required field

Name: *	<input type="text" value="Backup Schedule"/>	The name of this schedule.
Description:	<input type="text" value="Backup daily."/>	A description of this Schedule. This description will be visible in the list of scripts.
Build Interval (minutes): *	<input type="text" value="1440"/>	The amount of minutes that will pass before the scheduled task gets run.
Interval Start Time (hh:mm): *	<input type="text" value="23:30"/>	The time at which this schedule should start running.
Scheduled Actions		A list of the actions that are associated with this schedule

5. Click **Set**.

Select Backup Schedule (Derby)

1. Go to **System** > **Backup Settings** link under the Server menu.
2. To manually backup AnthillPro, click the **Backup Now** button.
3. To choose a schedule or change the number of stored backups, on the **Main** tab click the **Edit** button.
 - **Backup Schedule.** Select an existing schedule from the drop-down menu. See Create a Backup Schedule.
 - **Max Backups.** Enter the maximum number of backups that will be stored. Use -1 to keep all backups. If Max Backups is set to 25, AnthillPro will save the first 25 backups. When the 26th backup is created, AnthillPro will automatically delete the *oldest* backup.

If reducing the number of backups stored, AnthillPro will delete all backups in excess of the new setting. For example, if changing backup storage from 25 to 20, the next time a backup runs AnthillPro will automatically remove the 6 (six) oldest backups and add the new backup.

- Click **Save**.

Backup Settings for the Anthill3 server.

*denotes required field

Backup Schedule: * Select the schedule that the backup process will run on.

Max Backups: * Enter the maximum number of backups that will be stored. All others will be removed during the scheduled backup. Use -1 to keep all backups.

4. Select the **Backups** tab to view stored backups that can be restored or deleted. Backups are saved in the var\db directory (e.g., C:\AnthillPro\server\var\db\backup\). Every update will be named according to the date and time of the backup (in the format year-month-day_hour-minute-second; i.e., similar to 2008-10-08_10-59-48).

Name	Backup Date	Operations
2008-01-17 15:29:09	Thu Jan 17 15:29:09 EST 2008	X

5. Select the **History** tab for a list of database changes.

The screenshot shows a web interface with three tabs: 'Main', 'Backups', and 'History'. The 'History' tab is active, displaying a message 'List of the database change history.' above a table. The table has three columns: 'Description', 'User', and 'Date'. A single row of data is visible, showing a backup performed by 'admin' on 'Thu Jan 17 15:29:41 EST 2008'.

Description	User	Date
Backed up the database as backup 2008-01-17 15:29:09 at version 109	admin	Thu Jan 17 15:29:41 EST 2008

Deactivate or Change Backup Schedule (Derby)

When deactivating or changing a backup schedule, make sure that the schedule (created in Create a Backup Schedule) is not inadvertently deactivated or deleted. Because different AnthillPro activities may rely on a single schedule, deleting or deactivating the actual schedule (at **System > Schedules**) may effect other activities.

To deactivate or change the backup schedule:

1. Go to **System > Backup Settings** link under the Server menu.
2. Click **Edit** on the Backup Setting Main tab.
3. **To deactivate backups:** Select **None** from the Backup Schedule drop-down menu to deactivate backups. When none is selected, no further backups will occur.
4. **To change to a different schedule:** Select it from the drop-down menu. This may require creating a new schedule (see Create a Backup Schedule). One complete, a backup will occur when the new schedule fires.
5. Click **Save** then **Done**.

Restore AnthillPro to a Clean State (Derby)

Restoring AnthillPro to a clean state requires overlaying the installation with backed-up data files. The directions outlined below are for backups of the embedded Derby database (see Backups with Derby). If using another database type, you must follow the procedures listed in the Backups with Other Databases section. Typically, restoring AnthillPro to a backed-up version is an option of last resort, and should only be used if there is no other way to correct an issue.

Note that restoration requires shutting down the server and agent(s).

1. Shutdown the server and the agent(s).
2. Go to the AnthillPro instance.
3. In the `var\db\` directory, rename the **data** folder. In Item 7, a copy of the backed-up data folder will be added (renaming the existing **data** folder reduces the chances of corruption).
4. Go to the `var\db\backup\` directory and open the appropriate backup folder. Every backup is identified in the format year-month-date_hour-minute-second (e.g., 2008-10-08_10-59-48).
5. Open the backup you want to restore AnthillPro to, and copy the **data** folder located within the directory (e.g., the data folder is located in the `var\db\backup\2008-10-08_10-59-48\` directory of a backup that was performed on October 8, 2008 at 10:59:48 a.m.).
6. Go back to the `var\db\` directory.

7. Paste the backup's **data** folder into the `var\db\` directory. Do not delete the old (renamed) data folder until you have verified that the restoration was successful.
8. Restart the server and agent(s). Ensure that the restoration was successful. If there is a problem, please contact the AnthillPro support team with details.

Backups with Other Databases

If using AnthillPro with Oracle, MySQL, or Microsoft SQL Server database, backups are not performed through the AnthillPro UI. Each database has its own backup tool that should be used for backups (see documentation for your particular tool). *The var/db directory should never be backed up with a file system backup (it is the location of the database files) because this will almost always result in corrupted files.*

The following directories also need to be backed up using a file system backup tool:

Additional Directories to Backup (Non-derby)	
File	Contents
<code>conf/ah3.keystore</code>	Server keys for secure server/agent communication.
<code>opt/tomcat/conf/tomcat.keystore</code>	Server key for using HTTPS.
<code>var/artifacts</code>	Artifacts published on Build Lives.
<code>var/changelog</code>	Change logs published.
<code>var/codestation</code>	Codestation project artifacts.
<code>var/log</code>	All logs for requests, workflows, jobs, steps and commands.
<code>var/mavencache</code>	<i>Required only if using the Maven integration.</i>
<code>var/published</code>	Published reports on Build Lives
<code>var/reports</code>	Raw data reports used to create published reports.

- Because the Agent does not store critical data (other than agent configurations), Agent upgrade is unnecessary.

To use a backed up database, see Using a Backed Up Non-derby Database.

Using a Backed Up Non-derby Database

1. Perform a clean install of the AnthillPro server. See Installing AnthillPro.
2. If the server is running, stop the server. Go to the `\bin` directory and run `stop_ah3server.cmd` for Windows; or `./ah3server stop` for Linux/Unix.
3. Open the `installed.properties` file located in the `\conf\server` directory.
 - The `install.db.url=jdbc\:` property must be changed to point to the backed up database. For example, `install.db.url=jdbc\:yourdatabasetype\://localhost\:11366/data`.
 - Save change.
4. Open the `base.xml` file located in the `\conf\spring-server` directory. Modify two properties to point to the backed up database:

- Under `<!-- DataSource used for persistence -->`, modify the 'url' property:

```
<property name="url">
  <value>jdbc:yourdatabase://localhost:11366/data</value>
</property>
```

- Under `<!-- DataSource used for identity generation so it does not create deadlocks -->`, modify the 'url' property:

```
<property name="url">
  <value>jdbc:yourdatabase://localhost:11366/data</value>
</property>
```

- Depending on how the new installation was completed, the 'username' and 'password' properties may also need to be changed in both locations:

```
<property name="username">
  <value>anthill3</value>
</property>
```

```
<property name="password">
  <value>abc{}</value>
</property>
```

- Save changes.

5. Copy over the directories that were backed up using the file system backup tool. See [Additional Directories to Backup \(Non-derby\)](#).

6. Start the Server.

Chapter 52. Optimizing Server Performance

For advanced users experienced with web applications, there are a number of things you can do to increase performance of the AnthillPro server. Much of what is included in this section is based on current users who have successfully employed techniques to improve performance.

Caching and AnthillPro

To improve server performance, it is possible to set up a caching proxy (such as Squid [<http://www.squid-cache.org/>]) to reduce bandwidth and improve AnthillPro's response times. This is especially helpful in distributed development environments: the proxy can improve performance for users at off-site locations because commonly used pages are loaded from the locally stored cache. Configure the caching proxy with AnthillPro just like any other web application. However, if the AnthillPro server was set up to use SSL, see Caching and SSL before attempting to use a caching proxy.

Caching and SSL

To improve server performance, it is possible to set up a caching proxy (such as Squid [<http://www.squid-cache.org/>]) to reduce bandwidth and improve AnthillPro's response times. This is especially helpful in distributed development environments: the proxy can improve performance for users at off-site locations because commonly used pages are loaded from the locally stored cache. Unless the server uses SSL, configuring the caching proxy with AnthillPro is the same as any other web application. See Caching and AnthillPro.

However, if AnthillPro was set up to use SSL (see Installing AnthillPro), Tomcat prevents web pages from being cached. By default, Tomcat includes "pragma:no-cache" as an HTTP header whenever SSL is enabled. To remove this header, the security-constraints section of the web.xml file (located in the `\opt\tomcat\conf` directory) must be modified. See Item 3 below.

Once completed, this will enable use of the proxy, as well as allow Tomcat to use both HTTP and HTTPS connections. If communication must always use SSL, the server.xml file (located in the `\opt\tomcat\conf` directory) must also be modified. There are numerous approaches to control how SSL is used-- each dependent on the desired behavior, the tools used, the network architecture, etc. Check with the system administrator to decide how to handle SSL.

To configure a caching proxy with SSL:

1. Shut down the AnthillPro server and agent(s).
2. Open the **web.xml** file, located in the `\opt\tomcat\conf` directory, in a text editor.
3. Remove the security-constraint section (that includes the "pragma:no-cache" HTTP header) from the web.xml file. Typically, the header will be included under the `<!-- security-constraints -->` heading, and should be similar to:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>
      Automatic SSL Forwarding
    </web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
</security-constraint>
```



```
<transport-guarantee>
  CONFIDENTIAL
</transport-guarantee>
</user-data-constraint>
</security-constraint>
```

4. Save changes to the web.xml file.
5. Restart the AnthillPro server and agent(s). AnthillPro can now use the proxy, with Tomcat using both HTTP and HTTPS connections.

If communication must always use SSL, the **server.xml** file (located in the `\opt\tomcat\conf` directory) must be modified. For example, you can disable the HTTP connector, enable an AJP connector, and set up a HTTPD server that handles SSL connection. However, there are numerous approaches to control how SSL is used, so check with the system administrator to decide how to handle SSL.

Uploading Artifacts to a Build Life

Under certain conditions the server (Codestation) may throw the following error: `Server Error: HTTP Code 500: Could not acquire exclusive access to the buildlife`. If you see this error when Codestation is attempting to upload multiple artifacts to the same Build Life, the first upload most likely exceeds the 15-second default time-out. To address this issue, see Codestation, Artifact Sets, and Exclusive Locks.

Overriding Server and Agent File Storage Settings

The AnthillPro server and agent allow you to override the default location for the storage of a number of things (see Item Two below). If it is necessary to override the default locations, the startup script must be modified to pass a system property (see Item One below). Before overriding the default location(s), keep the following in mind:

- **Storage Directory.** If you change a storage directory, you are responsible for moving existing files in the directory to the new storage location.
- **Upgrading.** A server upgrade will overwrite the startup script and the setting will need to be added again.
- **Windows Service.** If running the server as a Windows service, the **JAVA_OPTS** (see below) in the service installer in the **bin/service** directory will need to be modified. Once modified, the service will need to be removed and reinstalled.
- **Windows Mapped Drive.** If running the server on Windows and using a mapped drive as a storage location, Windows has been known to remove the mapping almost at random due to inactivity. See Microsoft Support [<http://support.microsoft.com/kb/297684>].

To override the default storage location for the AnthillPro server and/or agent, modify the startup script to pass a system property:

1. Go to the AnthillPro **bin** directory and open the **ah3server.cmd** or **ah3server** file (the file name depends on what platform the server is installed on) in a text editor.
2. Find the **JAVA_OPTS** definition of the property you want to modify. See the appropriate property to be overrid-

den:

- Server or Agent Properties
- Server Properties
- Agent Properties

3. **Set a new system property.** The property needs to be prefixed with **-D** and a = (equals sign) should separate the property name from its value. For example, the default: `../myvar` for **anthill3.var.dir** would become: **-Danthill3.var.dir=../myvar**.

A relative path can be given as the value and the relative path is calculated from the server's bin directory. A absolute path is also an option.

Override Server or Agent Properties

Before making any changes, see Overriding Server and Agent File Storage Settings.

- **anthill3.var.dir** (defaults to `../var`). Not used to store files directly (see Overriding Agent Properties).
- **anthill3.logs.dir** (defaults to `../var/log`). Storage for logging files using the default logging configuration.
- **anthill3.temp.dir** (defaults to `../var/temp`). Storage for temporary files that created for short-term temporary use.

Override Server Properties

Before making any changes, see Overriding Server and Agent File Storage Settings.

- **anthill3.artifacts.dir** (defaults to `../var/artifacts`). Storage for files published to Codestation.
- **anthill3.codestation.dir** (defaults to `../var/codestation`). Storage for file in Codestation project Build Lives.
- **anthill3.publish.dir** (defaults to `../var/published`). Storage for published files that are accessible under the Reports tab of the Build Life.
- **anthill3.reports.dir** (defaults to `../var/reports`). Storage for published data files that are not accessible but used by the server.
- **anthill3.mavencache.dir** (defaults to `../var/mavencache`). Storage for files used with the Maven dependency integration.
- **anthill3.changelog.dir** (defaults to `../var/changelog`). Storage for data files used internally by AnthillPro for change log processing.

Override Agent Properties

Before making any changes, see Overriding Server and Agent File Storage Settings.

- **anthill3.work.dir** (defaults to `../work/jobs`). The base directory for agent working directories.

Server Logging Options

AnthillPro creates two log files: **ah3server.out** (which contains all logging messages) and **ah3server.err** (containing stack traces). Both logs are viewable through the UI at: **System > Server Settings > Log/Error**, or in the server's `\var\log` directory.

By default, the **ah3server.out** log has a maximum size of 50 MB, and the **ah3server.err** log has a maximum size of 5 MB. For either log, when the maximum file size is reached, the log will automatically roll over. To either increase or decrease the log size:

1. Shut down the AnthillPro server (run: **stop_ah3server.cmd** located in the server's `\bin` directory).
2. Go to the server's `\conf\server` directory and open the **log4j.properties** file in a text editor.
3. To change the main **server log** file size, find the **file appender** section and modify the following setting (usually on line 9):
 - `log4j.appender.file.MaxFileSize=50MB`
4. To change the main **error log** file size, find the **error file appender** section and modify the following setting (usually on line 17):
 - `log4j.appender.errfile.MaxFileSize=5MB`
5. Save changes when done.
6. Restart the server (run: **start_ah3server.cmd** or **run_ah3server.cmd** located in the server's `\bin` directory).

To change/set the logging level, see [Configure Server Miscellaneous Settings](#).

Server Diagnostics

The Diagnostics tab (at **System > Server Settings > Diagnostics**) provides useful information when debugging the system. For those with permissions to the System page (typically AnthillPro administrators), you can run reports on the active request contexts; thread CPU usage; the system properties; and any scheduled items while troubleshooting the system.

Chapter 53. Create Server Proxy

If you need to access servers/repositories (such as Maven, ibiblio, Apache/Jakarta, etc.) that are on external networks, do so by going to the System page and creating a new proxy. To help secure the connection, set a password-protected user name on the proxy. To get started, make sure:

- You must have AnthillPro administrative privileges to the System page. See Manage Security.
- The proxy host and port must be available.

To configure the proxy:

1. Go to **System > Proxy Settings** under the Server menu.
2. On the Proxy Settings page, click **Create Proxy**.
3. Set up Proxy:
 - **Proxy Name.** Give the name AnthillPro will use for the proxy.
 - **Description.** Give a description of the proxy (e.g., Maven proxy).
 - **Host.** Provide the host for the proxy.
 - **Port.** Give the port AnthillPro will use to communicate with the proxy.
 - **Username.** Provide the name to be used for proxy authentication. This is the name AnthillPro will use when connecting to the proxied server/repository (e.g., anthill3mavenproxy).
 - **Password.** Give the password (if password protected). This is the password used to authenticate the AnthillPro user name set above.
4. Click **Save** then **Done**.

Chapter 54. Clone AnthillPro Instance

Cloning an instance of your AnthillPro server will allow you to change scripts, optimize processes, improve build performance, move the server, etc., without having to experiment on the production server. The process involves installing a clean version of AnthillPro in a new location, cloning the production database, and copying over the cloned database to the new instance. If necessary, you can move the artifacts, logs, and reports to the new server.

If you plan on changing the database type, see [Migrate Server Database](#).

Before you begin, make sure:

- The anthill3 distributable either `anthill3-[version].zip` or `anthill3-[version].tar.gz` for the version of AnthillPro which is currently installed as the production instance. For example, if your production server is Version 3.7.0_52506, you will need to install Version 3.7.0_52506 in the new instance. Otherwise, you may not be able to copy over the database to the new instance. If you can't find the correct distributable, please contact support [<http://support.urbancode.com/>].
- Your evaluation license (different from the production server). To obtain a copy of the evaluation license, login to the AnthillPro Supportal [<http://support.urbancode.com/>].
- The ability to start and stop the production AnthillPro instance.
- Access to the AnthillPro production database.

Once the prerequisites are met, you can clone then move the database:

1. Install the new AnthillPro server and instruct it to connect to a new empty database created specifically for the instance.
2. Modify the production server to ensure that it is configured to bind to the address: 0.0.0.0 on startup. Go to **System > Server Settings**, and set the **Bind to IP** setting to 0.0.0.0.
3. Shutdown the AnthillPro production server.
4. Clone the AnthillPro database. The procedure varies depending on the underlying database. Your database vendor should have documentation on cloning.
5. Start the production AnthillPro server. If need be, reset the **Bind to IP** setting. This may require you to restart the server to take effect.
6. Copy the cloned database over the top of the clean database schema which was created in Item 1.
7. Cleanup the cloned database before installing it in the new instance.

The first two statements (see below) remove the production license from the new server to prevent license conflicts. If you do not run these statements, starting the new server may disable your production server.

The third statement instructs the new instance to ignore all production agents, preventing accidental upgrades.

Connect to the new database using a tool such as DBVisualizer (or the console) and issue the following SQL statements:

- `DELETE FROM ANTHILL_LICENSE;`

- DELETE FROM REPOSITORY_USERS;
 - UPDATE AGENT SET ISIGNORED = 1;
8. Moving SSL certificates. If your production server is using SSL, you will need to copy over all the *.keystore files into the new instance. Do not remove the files from the production server.
 9. Start the new AnthillPro instance and enter your evaluation license. In order to use the server, you will have to connect it to at least one agent. There are a number of options you can choose: the simplest is to install some new agents and bind them to the new server.

Note that if you have any production agents come online (e.g., you restart an agent) while the new instance is running and still set to bind to 0.0.0.0, that agent may bind to the new instance and not the production instance. To prevent accidental binding and/or upgrade, you can give the AnthillPro new instance a unique IP address and then tell its agents only to bind to that IP.

Move Artifacts, Logs, and Reports

If you need to move the artifacts, logs, reports, etc., stored on the server, you will need to copy the /var directory from the old server to the new instance. When moving, you must ensure that you exactly copy over the directory -- otherwise AnthillPro will not be able to properly retrieve artifacts, etc. To move the artifacts, etc.:

1. Ensure the new AnthillPro instance has been installed correctly and stop the server if it is running.
2. Go to the old AnthillPro instance and copy the entire /var directory. It is important to match the file system exactly!
3. Once copied, overwrite the /var directory of the new instance. This may take a while, depending on how many artifacts are stored on the machine.
4. When done, restart the new AnthillPro instance.

Chapter 55. Moving the AnthillPro Database

Before you begin, make sure you back up your existing database. Once that is done, you can then clone the database to the new location (it is recommended to retain the original database until you are sure the move was successful).

Before you begin, make sure you have:

- The ability to start and stop the production AnthillPro instance.
- Access to the AnthillPro server `\conf` directory.
- Access to the AnthillPro production database.

To move the AnthillPro database:

1. Shutdown the AnthillPro production server.
2. Clone the AnthillPro database and *install the clone in the desired location*. The procedure varies depending on the underlying database. Your database vendor should have documentation on cloning.
3. Go to the AnthillPro server and open the `installed.properties` file located in the `\conf\server` directory.

- The `install.db.url=jdbc\:` property must be changed to point to the new database location. For example, `install.db.url=jdbc\:yourdatabasetype\://localhost\:11366/data`.

Note that the property for each database type is different and may have a different pattern. If need be, check with your database administrator.

- Save change.
4. Open the `base.xml` file located in the `\conf\spring-server` directory. Modify two properties to point to the new database location:

- Under `<!-- DataSource used for persistence -->`, modify the 'url' property:

```
<property name="url">
  <value>jdbc:yourdatabase://localhost:11366/data</value>
</property>
```

- Under `<!-- DataSource used for identity generation so it does not create deadlocks -->`, modify the 'url' property:

```
<property name="url">
  <value>jdbc:yourdatabase://localhost:11366/data</value>
</property>
```

- Save changes.
5. Start the Server.

Chapter 56. Distributed Servers

Distributed Servers is a complimentary product to AnthillPro designed to reduce the reliance on WAN connections by performing all the heavy lifting (builds, etc.) on local networks. There are three main components to Distributed Servers:

- **Distributed Web Interface.** The interface mirrors the AnthillPro server, providing the same information and functionality as the main server, for AnthillPro users.
- **Codestation 2.0.** Allows off-site instances of the AnthillPro Distributed Server to locally store artifacts and other large data files. Codestation is included as part of the Distributed Web Interface installation.
- **Agent Relay (optional).** The new communication relay acts as a proxy for agents (which perform work on behalf of the AnthillPro server) that are located behind a firewall or in another network location. Available under separate download.

When using Distributed Servers, the main AnthillPro server is first installed at the central location; the Distributed Web UI is installed at the off-site locations (i.e., if the main server is installed in New York, the Distributed Web UI could be installed in Boston, etc.); and then the Agent Relay (if being used) is installed at off-site locations. Finally, agents are installed. See [Installing Distributed Servers](#).

Once installed, Distributed Servers are managed from within the AnthillPro UI on the System page. See [Managing Distributed Servers](#).

While there are numerous ways to use Distributed Servers, there are a few recommended best practices. See [Best Practices for using Distributed Servers](#).

Upgrading Distributed Servers

Beginning with version 3.6.3, AnthillPro provides an upgrade path for Distributed Servers. If you are using an older version (i.e., 3.6.2 or older), you will need to install version 3.6.3 in order to perform future upgrades.

Upgrade Distributed Web Interface

1. Before performing the upgrade, stop the Distributed Servers and backup the Distributed Servers directory.
2. Run the install script in the Distributed Servers installation package.
3. When asked for the directory, give the directory the old version is installed in.
4. The installer will ask you if you want to upgrade. Answer **Y**.
5. Follow the instructions for an installation.

Upgrade Agent Relay

1. Before performing the upgrade, stop the Agent Relay and backup the Agent Relay directory.
2. Run the install script in the Agent Relay installation package.

3. When asked for the directory, give the directory the old version is installed in.
4. The installer will ask you if you want to upgrade. Answer **Y**.
5. Follow the instructions for an installation.

Managing Distributed Servers

Management of Distributed Servers is performed on the **System > Distributed Servers** link under the Server menu.

On the Distributed Servers List page, the main AnthillPro server and any Distributed Servers are listed. The Active column indicates what servers are active. To activate an instance of Distributed Servers, or to edit settings, select the Edit under the Operations column. See Activate Distributed Servers and Edit Distributed Servers Settings.

Name	Description	Address	Active	Operations
AnthillPro3-Main		http://0.0.0.0:8080	●	
dist-web		http://localhost:8080	●	

Prerequisites: Managing Distributed Servers

- The AnthillPro server, Distributed Web Interface, and Agent Relay (if in use) must already be installed. See Installing AnthillPro and Installing Distributed Servers.
- You must have access to the main AnthillPro server, including permissions to manage Distributed Servers.

Activate Distributed Servers

Once the Distributed Web Interface has been installed (see Installing Distributed Servers), it will appear on the Distributed Server List page (go to **System > Distributed Servers** to view). If you can't access the System page, that means you do not have the appropriate permissions. Please contact your AnthillPro administration if you need to activate a Distributed Server.

By default, only the user with System Administration role may activate a Distributed Web Interface.

To activate an instance of Distributed Servers:

1. Go to the main AnthillPro server.
2. On the **System** page, select the **Distributed Servers** link under the Server menu.
3. Select the Edit icon under the Operations menu of the Distributed Server to be activated. Inactive Distributed Servers are indicated by a brown icon under the Active menu.
4. When the Edit dialog opens, check the **Active** box. If you are unable to check the box, that means you do not have the appropriate permissions to activate the Distributed Server. Please contact your AnthillPro administrator.
5. By default, the **Accessible by All** box is checked. This allows all AnthillPro users to log into the distributed server, regardless of the access permissions (see Edit Distributed Server Settings).

6. Click **Save** if not setting permissions. Otherwise, see Edit Distributed Servers Settings.

Edit Distributed Servers Settings

1. Go to the main AnthillPro server.
2. On the **System** page, select the **Distributed Servers** link under the Server menu.
3. Select the edit icon under the Operations menu of the Distributed Server that permissions are being set for.

If the Distributed Server is inactive, see Activate Distributed Servers.
4. When the Edit dialog opens, assign permission to the appropriate Roles (see Distributed Servers: Best Practices for Setting Up Users).

Note that the Accessible by All option overrides any permissions set below. See Activate Distributed Servers.
 - **Manage.** Determine which roles are able to manage this Distributed Server instance and set security.
 - **View.** Select the roles than can view this Distributed Server instance on the list page.
 - **Access.** Determine which roles can log into this Distributed Server instance.
5. Click **Save**.

Best Practices for Using Distributed Servers

While Distributed Servers can be used in a number of ways, the following best practices will help guide you when setting up and using AnthillPro in a distributed environment:

- **Installation.** There are a number of options for installation. This allows AnthillPro to fit into your organizational structures. See Distributed Servers: Best Practices for Installation.
- **Projects.** Model projects to geographic location. Set up projects to build, deploy, etc., within the same LAN, or geographic location. See Distributed Servers: Best Practices for Project Set Up.
- **Source Control.** To get the most out of Distributed Servers, the source should be stored within the same network as the Distributed Web Interface. See Distributed Servers: Best Practices for Source Management.
- **Users.** Typically, users are restricted to perform builds, deployments, etc., from within the same LAN as the Distributed Web Interface they use. See Distributed Servers: Best Practices for Setting Up Users.

Distributed Servers: Best Practices for Installation

While there is no particular order to installing Distributed Servers, the following sequence is typically followed:

1. Install the **AnthillPro main server**. See Installing AnthillPro.
2. Install the **Distributed Web Interface**. The interface is typically installed on a different LAN than the server. This enables administrative personnel to isolate the work performed by teams using the Distributed Web Interface, and restrict the actions users perform (such as builds, deployments, etc.) to the LAN they are assigned to.

The interface may be installed on either a Windows or Linux/Unix machine, regardless of the OS used when installing the main server (e.g., the AnthillPro server may be installed on a Windows machine and the Distributed Web Interface may be installed on a Linux/Unix machine and vice versa). See either [Install Distributed Web Interface \(Windows\)](#) or [Install Distributed Web Interface \(Linux/Unix\)](#).

3. Install the **Agent Relay**. The relay is typically used when agents are to be installed behind a firewall or in another network location. If an instance of the Distributed Web Interface requires special hardware or software housed outside of the LAN, the relay allows agents to connect to the main server using a single port. So, any agent installed anywhere in the AnthillPro system (no matter of location) will use a single port (the relay) when contacting the central server. This makes it easier when dealing with firewalls and connecting between networks.

The Relay may be installed on either a Windows or Linux/Unix machine, regardless of the OS used when installing the main server and Distributed Web Interface (e.g., the AnthillPro server may be installed on a Windows machine; the Distributed Web Interface may be installed on a Linux/Unix machine; and the Relay may be installed on a Windows machine, etc.). See either [Install Agent Relay \(Windows\)](#) or [Install Agent Relay \(Linux/Unix\)](#).

4. Install **Agents**. Typically, install the Agents on the same network as the Distributed Web Interface, and configure them to receive work only from that server. This will result in AnthillPro running builds, etc., within the same LAN. Any number of agents may be installed along with the Distributed Web Interface, creating a "virtual" server that works "independently" from the main server. However, depending on your organizations distributed network, some agents on other networks may be necessary to complete a process (e.g., to build a project that requires special hardware, etc.). See [Distributed Servers: Best Practices for Project Set Up](#).

The Agent(s) may be installed on either a Windows or Linux/Unix machine, regardless of the OS used when installing the main server; Distributed Web Interface, or Agent Relay (e.g., the AnthillPro server may be installed on a Windows machine; the Distributed Web Interface may be installed on a Linux/Unix machine; the Relay may be installed on a Windows machine; and the Agent may be installed on a Unix machine, etc.).

Once an agent is installed, it must be configured with the appropriate Distributed Server, which is a simple process. See [Distributed Servers and Agent Configuration](#).

Distributed Servers: Best Practices for Project Set Up

With Distributed Servers, AnthillPro projects are set up in the same way as before; however, to maximize the use of Distributed Servers, there are some helpful tips to keep in mind:

- **Source Code.** Whenever possible, create a project that pulls source code from a repository located on the same LAN as the Distributed Web Interface. See [Distributed Servers: Best Practices for Source Management](#).
- **Builders and other tools.** As with Source Code, install all the builders, testing tools, and other software used to build, test and deploy a project within the same LAN as the Distributed Web Interface. This will help isolate the project within one LAN.
- **Users.** Restrict users to use an instance of the Distributed Web Interface installed on the LAN they are a part of. This will ensure that users can only perform actions within their LAN, and reduce reliance on WAN connections. See [Distributed Servers: Best Practices for Setting Up Users](#).
- **Agent Selection.** When setting up a project to use agents, select agents that are installed on the same LAN as the Distributed Web Interface, and then ensure that users that can access the agent are also on the same LAN. However, if there is a situation where an agent outside of the local network is necessary (e.g., if special hardware is needed) to perform the process, try decomposing the job so that as few steps as possible are run on the remote agent. This will minimize WAN usage.
- **Agent Relay.** For projects that must be built on an agent located behind a firewall or on a different LAN, use the

Agent Relay. With the relay, only one port is used for all communication between agents and the central server.

Distributed Servers: Best Practices for Source Management

It is recommended that the source code used for builds be stored in the same LAN as the Distributed Web Interface. When this is done, builds will be isolated to the LAN in which the Distributed Web Interface is installed. All work will then be performed within the same LAN, without having to open a WAN connection to the central server.

In order to accomplish this, it may be necessary to install another version of your SCM at the off-site location (i.e., within the same LAN as the Distributed Web Interface) that contains the appropriate source. The source stored at the off-site location should mirror the projects that team members use when accessing the Distributed Web Interface. For example, if a team in Boston works on Project A, set up a repository in Boston that contains Project A. When team members call for a build, the source will be taken from the local repository and work will be performed using LAN connections.

Dependency management is automated when using Distributed Servers, so if a dependency is required -- say a project located in a different LAN -- AnthillPro will automatically open a WAN connection to resolve the dependency as part of the build, etc. Once the dependency is satisfied, the build, etc., will take place within the LAN that the request originated. For example, if a build of Project A, with its source in Boston, is dependent on a project with source located in New York, AnthillPro will automatically open a WAN connection to the New York server to resolve the dependency. Then, the build will be performed on the Boston instance of AnthillPro.

Distributed Servers: Best Practices for Setting Up Users

Once the roles have been assigned for the Distributed Server (see Edit Distributed Servers Settings), assign users to roles using the AnthillPro security system (see Add Users). As a best practice, restrict users to perform work within the same LAN as the Distributed Web Interface they use.

This may be accomplished by either creating a specific role for users (see Define Roles), based on their geographic location, or my determining permissions on the Distributed Servers list page (see Edit Distributed Servers Settings).

Chapter 57. Using Externally Signed Certificate with Tomcat

If you have a third-party certificate (e.g., signed by Equifax, etc.) you would like to use, you can include it in the AnthillPro tomcat.keystore. This will allow you to use the signed certificate without HTTPD. To use a third-party signed key:

1. Get the CA public certificate from your vendor (this is the public portion of the key that they used for signing. It should be a ca.crt file).
2. Generate a key which will be signed with the "ca.key" file. The output of this step is the "server.key" file which is the private portion of your SSL key and the "server.crt" file which is the signed certificate.
3. Export the CA public key, the server private key, and signed certificate into a single PKCS12 keystore.
4. Edit the \$AH3_SERVER/opt/tomcat/conf/server.xml file. Copy the generated tomcat.keystore from the last step into the '\$AH3_SERVER/opt/tomcat/conf' directory. Then edit the \$AH3_SERVER/opt/tomcat/conf/server.xml file. The XML element <Connector port="8443" ...> needs to contain the attribute keystorePass="password associated with your exported keystore" keystoreType="PKCS12".
5. Restart the server. AnthillPro should be using your signed certificate.

Part XI. Agent Management

AnthillPro agents are light-weight Java processes that are installed on machines throughout your networks. Once installed on a machine, the agent allows AnthillPro to run commands on the machine, move files to and from the machine, and work with other tools installed on the machine.

In general, an agent is responsible for running builds, deploying projects, and/or driving third-party tools. When installing agents, keeping in mind what processes you want AnthillPro to perform, and where you want those processes to take place, can be a helpful guide. Following is a quick synopsis of the most common agent-usage scenarios and where agents are typically installed:

- **For builds.** Agents are typically installed on every build machine. This allows you to maximize resources and enables simultaneous builds on different agents. If you have a project that requires a specific tool or platform to build, you can use, for example, an agent filter to specify which machine or set of machines to use.
- **For deployments.** Many users install an agent on the deployment target machines -- particularly when a deployment requires running commands on a specific machine. Where a remote deployment is run, an agent is often placed on a machine that has access to the various target machines. In either case, the agent acts as a gateway for deployments.
- **For driving third-party tools.** AnthillPro integrates with numerous tools and normally communicates with those tools through its agents. AnthillPro agents are responsible for triggering automated testing, creating an issue in your tracking tool, etc., on behalf of the server. In some instances, the agent must be installed on the same machine as the tool it is driving (where necessary, this is noted in the user documentation). Most third-party tools can be driven remotely through web services or API calls. The agent calling those tools would simply need to have access to those web services or APIs.

Once an installed agent has been started, the agent opens a socket connection to the server (securable by configuring SSL for server-agent communication) based on the information you gave during installation. The agent also utilizes AnthillPro's web services. For agents on a different network than the server, it may be necessary to punch some holes in a firewall to establish the connection. If the number of agents needing firewall exceptions grows inconveniently large, the Agent Relay can help. Once agent communication is established, the agent will be visible through the web UI (**Agents > Agent**), where it will need to be configured.

If you are unfamiliar with AnthillPro Environments, it may be helpful to review that material to gain a better understanding of how AnthillPro Agents are managed and used. In short, Agents are pooled into environments. In AnthillPro, environments denote similar, high-level purposes of machines, often corresponding to groupings like "Shared Build Farm" or "My Department QA Lab." In turn, environments are pooled into groups usually corresponding to the different applications under development that use different sets of machines. Understanding how AnthillPro approaches environment management is necessary when configuring an agent.

Agent configuration consists of assigning an agent to at least one environment (as well as adding properties and entering the load-balancing rules when appropriate). An agent not part of an environment will have few, if any, jobs assigned to it. AnthillPro also allows you to assign an agent to multiple environments. During agent configuration, the system administrator defines which environments are legal for an agent. For example, if an agent is assigned to both the DEV and QA environments, the agent can perform work on behalf of projects in either environment -- this allows you to better distribute workloads throughout the system. When performing work, AnthillPro uses the administrator configured agent filters in conjunction with load balancing to assign jobs to appropriate agents within the environment.

In addition to system-created meta-data, every AnthillPro Agent may include properties assigned by an administrator. These properties are typically used to denote capabilities. Many users set a property that denotes if and where a particular tool is installed (say Visual Studio); or, the dedicated purpose of the machine (web server or database)

server). This meta-information can be used to help determine what work is performed on which agent. For example, if a deployment to the QA lab involves both web servers and database servers with different processes run on each, the user-defined agent properties are used to ensure that the deployment is run successfully.

For those with a large number of agents, you can use the **Filter** and **Status** fields to make searching for agents easier. Once the search is complete, you can then perform an Action on the returned list by selecting the appropriate agent(s).

About Agent Filters

Agent filters are used to select the appropriate agent(s) that will execute a job. Agent filters are scoped by environments, so they can only select an agent within the environments available to a workflow. For example, if you want a build to run on an agent within the "DEV" environment, you would use the Any Agent Filter. When the job is performed, AnthillPro will assign the build to the least busy agent in that environment. See Configure and Edit Agent Filters.

There are three different types of agent filters in AnthillPro:

- **Any Agent Filter** returns all online agents in the environment. The agents returned are ordered by a combination of their current load and throughput metric, and then the server selects the agent(s) to use based on their loads.
- **Fixed Agent Filter** returns a specified agent. Use of this agent filter is discouraged because the job will fail if the agent is not available in the selected environment. However, for new users, the fixed agent filter can help with getting started if it takes the place of a more elegant, but more time consuming and difficult to create, Scripted Agent Filter.
- **Scripted Agent Filter** programmatically selects the agent(s) appropriate for a job. Usually this is used to check variables on agents. For example, to find agents with a specific tool installed or used for a special purpose. When executed, the filter receives a collection of agents and is responsible for trimming that list to only appropriate agents. An example of Scripted Agent Filter usage can be seen in the Iterate a Job section.

About Agent Properties and Environment Variables

Agent properties allow the user to configure a build or other process using parameters with values dependent on the agent on which the build (or process) is actually executed. For example, an agent property can be used to specify that an agent's local settings be used during a build, etc. One thing to keep in mind is that agent properties are resolved at run time (i.e., when the build is actually run), and on the Agent on which the build is running. See Agent Properties for more.

Implicit and User Properties

Implicit properties have a designation indicating where they come from. All environment properties as well as all Java system properties on an agent are automatically loaded and made available as agent properties. You can add new agent properties through the user interface on the central server. These properties are immediately usable, just like the implicit environment and system properties.

Using Agent Properties

Consider the situation where you are configuring a build and need to use a specific version of Ant. You have a Linux machine and a Windows machine, both with AnthillPro agents, that could end up running the build. Further, Ant is installed in different locations on the Linux and Windows machines. You need to use an agent property to

configure the build.

Using agent properties, you can define a variable named `user/ANT_HOME_1_6_5` on both agents. On the Windows agent, the value of the property would be `user/ANT_HOME_1_6_5 = C:\apache\apache-ant-1.6.5`. On the Linux agent, the value of the property would be `user/ANT_HOME_1_6_5 = /opt/apache/apache-ant-1.6.5`.

By configuring the AnthillPro agent property as above, either agent would be able to perform the job. The job itself would use a special syntax instruct AnthillPro to run the Ant located at property: `${user/ANT_HOME_1_6_5}`.

Agent Environment Variables

Agent Environment Variables (not to be confused with AnthillPro Environment Properties) are set at the time of agent installation, and are stored on the central server. The agent-installation reads the environment variables of the machine the agent is being installed on and then sets the variables. Once set, AnthillPro will not check to see if any of the environment variables have changed. If you change one of the environment variables that the agent uses, it may cause the agent to fail. For example, if you change the `env/Path` variable on the machine, the agent will continue to use the original setting. In this case, you may need to modify the agent's environment variable to reflect the changes you made in order for the agent to work. This can be done on the Agent's **Environment** tab by clicking the Edit icon of the variable and making changes.

About Agent Throughput Metric

When configuring the central server to use an agent, you can assign a throughput metric. The throughput metric is an arbitrary integer indicating the relative throughput of one agent relative to another. For example, if one agent is running on a four CPU machine and a second is running on a single CPU machine, then the throughput metric of the first agent may be four and the throughput metric of the second agent may be one. AnthillPro considers current job loads and throughput when delegating jobs to agents.

Administrators configuring Agents can also set an absolute maximum number of jobs a particular agent can execute. If all agents selected by a filter are at their maximum, the next job will enter a priority queue waiting for ability.

Consider two agents. The first, "Quad" has a throughput of 9 and a max jobs setting of 6. The second, "Dual" has a throughput of 5 and a max of 4. If eleven jobs are generated quickly, they will be assigned something like:

Job 01	Quad
Job 02	Dual
Job 03	Quad
Job 04	Quad
Job 05	Dual
Job 06	Quad
Job 07	Quad
Job 08	Dual
Job 09	Quad
Job 10	Dual: (Quad is maxed)
Job 11	Queued (Quad and Dual are maxed)

About Agent Security

User access to an agent is managed on the Security tab, as well as assigning permissions using AnthillPro's security

system. Administrators can define what roles have access to read, write, or determine security for agents. You need administrative permissions to set agent security.

Chapter 58. Agent Configuration

To configure an agent, click on the name of that agent or the View link in any of the tables on the Agent Overview page. Most of the information shown will be configured at installation time, but you can edit the agent any time after installation. The Properties tab on the agent configuration screen allows you to view or set custom variables on the agent. The custom properties can indicate where build or testing tools are installed. In the Locked Properties section, you can review the system and environment variables (often used in agent filters).

Once configured, access to the agent can be restricted using the AnthillPro security features. Once security is set, only users with the appropriate roles and permissions will be able to use the agent for running builds, deployments, etc. See Setting Up Security.

See also About AnthillPro Agents.

Agent Configuration Prerequisites

- You must have administrative permissions. See Setting Up Security.
- If an AnthillPro Agent has not been installed, see Installing AnthillPro before continuing.
- Go to **Agents > Agent**. Once the agents are installed, ensure that they all are online.
- If the agent to be configured is offline (see Agent Settings), run **start_ah3agent.cmd** found in the agent's `\bin` directory to start the agent.
- At least one Environment must be active in order to configure the agent. By default, each agent is available to the Build-Farm; however, the agent does not have to participate in this environment.

Configuring an Agent

1. Go to **Agents > Agent**.
2. Select the **View** icon of agent to be configured under the **Available Agents** menu.
3. Most of the information on this tab was given during agent installation (See Installing AnthillPro); however, you will need to select the environment(s) this agent is going to participate in, as well as the preferred server if using distributed servers.
 - **Name.** Give the Agent a descriptive name. The current name of the agent was given during the installation process. AnthillPro will automatically update the agent if it's name is changed.
 - **Throughput Metric.** The throughput metric provides a hint to AnthillPro's load balancer. In a busy build farm, it may be that several machines could handle a request, but each is already running builds.

To determine which machine is best equipped to handle the additional build, the load balancer compares the machine's throughput metric number to the number of jobs that are running on it. So an agent with a metric of 10 will likely get a third job assigned to it before an agent with a metric of 1 gets its second job.

- **Maximum Number of Jobs.** Agents can also be given a maximum number of jobs that they may run. If an agent is running at full capacity, additional job requests are queued until the agent frees up.
- **Preferred Server.** If utilizing distributed servers, select the appropriate server. In most cases, the agent should be connected to the preferred server via a LAN. Otherwise, select "none". See Distributed Servers and Agent

Configuration.

- **Environments.** Check the box for the environment or environments this agent is to be in. If the environment you want to add this agent to is not shown, that means the environment has not yet been created. See Environment Management.

An agent can participate in one or more environment. Generally an agent belongs to a single environment, such as Build-Farm or Development Test or QA. But often a single agent is used as a staging sever for several environments, and so it should participant within those environments.

4. Click **Set** if setting properties and/or security. Otherwise, click **Set** then **Done**.
5. Select the **Properties** tab and click **Add Property** or **Add Secure Property** button. Either property type may be used, with the only difference being that a Secure Property will obfuscate the property in logs, etc. The process for creating either property type is the same:
 - **Property Name.** Give the name of the property to be used. This will be visible to anyone who can access this property.
 - **Property Value.** Give the value of the property. If using a secure property, verify the value which will be obfuscated throughout the system.

Custom properties typically indicate where build or testing tools are installed.

6. Click **Add** then **Save**. Click **Done** twice if not setting security. Otherwise, click the **Security** tab.
7. Click **Edit** and check the appropriate **Permissions** and **User Roles** available to this agent. See Setting Up Security.
8. Click **Done** twice.

Chapter 59. Environment Management

In AnthillPro, an environment is a partition grid of agents that is specific for different stages of a project life-cycle (QA, PROD, etc.). Environments may also be configured to a specific technology (Java, .NET, etc.).

Workflows within AnthillPro execute on a specified environment. This allows the definition of a single deployment workflow that can then be used to deploy the application to numerous environments.

For example, common environments are development (DEV), quality assurance (QA), and production (PROD). The DEV environment should include the set of AnthillPro agents that are specific to development. Once the DEV environment is created, it is added to an environment group (see Using Environment Groups) and then the AnthillPro Agents are added to the groups (see also Agent Settings).

AnthillPro has a default environment, the Build Farm, that contains all agents used in conjunction with originating workflows. Each build will be assigned to an agent in the Build Farm unless the project is configured to build in a specific environment. The Build Farm may also contain any other agent you choose (e.g., a deployment agent). In AnthillPro, the build farm is an implied environment and can't be deleted. Because there are no restrictions on how the Build-Farm is used, it may be used for deployments as well.

See also About AnthillPro Agents.

Environment Prerequisites

- You must have administrative permissions. See Setting Up Security.
- If an AnthillPro agent has not been installed, see Installing AnthillPro before continuing.
- Go to **Agents > Agent**. Once the agents are installed, ensure that they all are online. See Agent Settings.

Configuring an Environment

AnthillPro environments are typically modeled after the different stages of the application life-cycle (or technologies) your organization uses.

1. Go to **Agents > Environment**.
2. Click **Create Environment**.
3. Configure Environment:
 - **Name** the environment.
 - **Short Name**. Give the name to be used as a key in property files. For example, the "development" environment will be identified by the short name PROD in property files and on the Dashboard.
 - **Description**. Give an optional description of the environment.
4. Click **Save**.
5. **Set Environment Property (optional)**. The property set here will be used unless it is overridden by other properties. It is also possible to set a property in the environment for specific projects.
 - Select the **Add Property** link in the Environment Properties header.

- **Secure.** To make the property secure, check the box.
 - To add an additional property, select the **Add Property** link again.
 - **Click Save.**
6. Select the environment's **Security** tab and click **Edit**.
 7. Check the **Roles** and **permissions** for this environment. See Define Roles and Using Permissions.
 8. Click **Save** then click **Done** twice.

Editing an Environment

Once created, the name, short name and/or description of an environment may be edited. Editing an environment will only effect future usage of the environment.

1. Go to **Agents > Environment**.
2. Select the environment to be edited from the Environment List.
3. Click **Edit** on the Main tab.
4. Edit the name, short name and/or description.
5. To remove an agent from the environment, select the **Remove from Environment** icon under the Operations menu. If the tool tip says Remove from Environment (disabled) the agent may not be removed from the environment in this manner. To remove an agent from an environment, see Agent Settings.
6. Click **Save** then **Done**.
7. Click **Done**.

Removing an Environment

To remove an environment, ensure that it is not in use by any project. If the environment is in use, it must first be removed from every environment group that uses it. Removing an environment that is currently in use may require you to reconfigure other environment groups, projects, and workflows that depend on the environment.

1. Go to **Agents > Environment**.
2. If the Delete icon (under the Operations menu) of the environment to be removed is red, click the icon.

If the Delete icon is gray, the environment must be removed from all environment groups it has been added to (see Using Environment Groups). Once the environment has been removed and the Delete icon is red, click the icon.
3. Click **OK** in the dialog box.
4. Click **Done**.

Modifying Agent Settings (Environments)

Once an agent has been configured (see Agent Configuration), you can access/edit its settings from the Environments page. Using the Operations icons, you can either change settings or remove an agent from any environment (in addition to performing the same actions on the Agents page).

1. Go to **Agents > Environment**, and select an environment from the list.
2. On the Environment page, find the agent you wish to modify/remove from this environment (you may have to scroll down).
3. To remove the agent from this environment, select the **Remove from Environment** icon under the Operations menu.

Note that removing an agent from an environment will effect all projects that participate within that environment. For example, if multiple projects are configured to use a particular agent (e.g., one on a Linux machine) and that agent is removed, the workflows will fail unless another suitable agent is configured within the environment.

4. To edit an agent's settings, select the agent's Name or the View icon under the Operations menu.

Note that changing agent settings will effect all projects that use the agent. For example, if multiple projects are configured to use a particular agent (e.g., one on a Linux machine) based on a property, and the property is modified, the workflows will fail unless another suitable agent is configured.

- To change agent settings, see Agent Configuration.

5. After modifying agent settings, click **Done** to return to the Environment page.

Environment Security

User access to an environment is managed on the **Security** tab. Administrators can define what roles have access to read, write, or determine security for environments. You need administrative permissions to set environment security.

Using Environment Groups

Environment groups determine the set of environments that project workflows may be executed on. Each environment group must contain at least one environment, and is set at the Project level. For example, common environments are development (DEV), quality assurance (QA), and production (PROD). Once the environments for the stages of the life-cycle are created (see Environment Management), they are grouped together. AnthillPro will then use the environment group when determining which agent to send work to.

AnthillPro's **Default Environment Group** (the name may be different depending on what version of AnthillPro you use) contains all the default environments. Environments may be added to or deleted from this group.

Generally, different applications are likely to use different sets of machines for their build, deploy, test activity, and will likely use different environment groups. When you create a deployment workflow for instance, it is best practice to present the user with the environments that are actually relevant to the project (maybe 5 or 10) rather than all the ones in the system (potentially hundreds). Environment groups help a project administrator by restricting what other users can see. Environment groups may be formed per project, per team or per department depending on where hardware is shared. An environment may exist in multiple groups which would allow a shared build cluster to be in environment groups for projects that do not share test hardware.

Consider the following: An organization that develops software using two different technologies. The first is J2EE. Our hypothetical organization has a development, a QA, and a production J2EE environment, with each containing

application servers configured to the corporate standard. The second technology used is C++, with development, QA, and production environments for all C++ projects. The C++ environments use different physical servers than the corresponding J2EE environments. In this scenario, J2EE projects and C++ projects are configured into two environment groups: one called 'J2EE Env Group', and the other called 'C++ Env Group'. The 'J2EE Env Group' contains the J2EE development, QA, and production environment groups, and the 'C++ Env Group' contains the C++ environment groups. With these environment groups, the organization derailed any confusion about what set of servers a J2EE or C++ project should be deployed to.

See also About AnthillPro Agents.

Environment Groups Prerequisites

- You must have administrative permissions. See Setting Up Security.
- If an AnthillPro agent has not been installed, see Installing AnthillPro before continuing.
- Go to **Agents > Agent**. Once the agents are installed, ensure that they all are online. See Agent Settings.
- The appropriate Environments must be created. See Environment Management.

Configuring an Environment Group

1. Go to **Agents > Environment Group**.
2. Click the **Create Environment Group** button.
3. Configure environment group:
 - **Name** the environment group.
 - **Description**. Give an optional description of the environment.
 - Click **Save**.
4. Click the **Add an Environment** button, select an environment from the drop-down menu. See Environment Management.
5. Click **Add Environment** and then **Save**.
6. Repeat Items 2 through 5 for every environment to be added to the group.
7. Select the environment group's **Security** tab and click **Edit**.
8. Check the **Roles** and **permissions** for this environment group. See Define Roles and Using Permissions.
9. Click **Save** then click **Done** twice.

Editing an Environment Group

Once created, the name of, description of, and/or environments within an environment group may be edited. Editing an environment group will only effect future usage of the environment groups.

1. Go to **Agents > Environment Group**.

2. Select the environment group to be edited from the Environment Group List.
3. Click **Edit** on the Main tab.
4. Edit the name and/or description.
5. To remove an environment from the group, select the **Remove from Group** icon under the Operations menu.
6. Click **OK** in the dialog box.
7. To edit security, see Configuring an Environment, Items 6 through 8.
8. Click **Done** twice.

Removing an Environment Group

To remove an environment group, ensure that it is not in use by any project. If the environment group is in use, it must first be removed from any projects using it (listed under the Used in Projects menu). Removing an environment group that is currently in use may require you to reconfigure other environment groups, projects, and workflows that depend on the environment group.

1. Go to **Agents > Environment Group**.
2. If the Delete icon (under the Operations menu) of the environment group to be removed is red, click the icon.

If the Delete icon is gray, the environment group must be removed from any project using it. Once the environment has been removed and the Delete icon is red, click the icon.
3. Click **OK** in the dialog box.
4. Click **Done**.

Chapter 60. Configure and Edit Agent Filters

Agent filters are used to select one or more agent(s) at run time and to monitor the quiet period (see Using Agent Filters and Quiet Periods). Agent filters are scoped by environments, and select from the agents allocated to the environment. When an agent filter is asked to select one or more agent(s), it selects from the agents available in the environment within which the filter is executing.

Only scripted agent filters are capable of targeting multiple agents. See Agent Settings and Agent Filter (Selection) Scripts.

AnthillPro uses three types of agent filters:

- **Any Agent Filter.** Selects the first available agent. Returns all online agents in the environment, ordered by a combination of current load and throughput metric.
- **Fixed Agent Filter.** Always selects a specific agent within the environment. If the requested agent is locked or can't receive more work, the request will be queued until the agent frees up.
- **Scripted Agent Filter.** Selects agent based on an agent filter script. See Agent Filter (Selection) Scripts.

See also About AnthillPro Agents.

Using Agent Filters and Quiet Periods

Quiet periods are configured on the project (Administration page), and play an integral part in ensuring that the source code AnthillPro obtains from the SCM contains a consistent set of changes. The quiet period is a measure of time that must pass without anyone committing changes to the source. When the specified time has passed without commits, AnthillPro triggers a build.

Any agent-filter type may be used to monitor the quiet period. Unless a fixed agent filter is used, the selection will be from within the requested environment for the build.

1. Go to **Administration > Edit Project**.
2. Select the **Quiet Period** tab on the Project **Main** page.
 - **Quite Period.** Input the desired quite period in seconds. For CI, it is usually best practice to set the quiet period at 60 seconds. This usually allows a developer enough time to make all their related commits. If you find that you are seeing broken builds because related commits are not within the same build, try lengthening the quiet period.
 - **Quiet Period Type.** Select the quiet period type.
 - **Changelog Triggered Quiet Period.** Periodically polls the repository for changes since the last build; typically used with a scheduled trigger. The changelog quiet period should be used for any project that has at least one workflow using a scheduled trigger. For example, if your project has one workflow that uses a repository trigger (that builds every time a commit is made) and one workflow that uses a scheduled trigger for a nightly build, use the changelog quiet period. This will ensure that AnthillPro builds when you want it to. See also Workflow Triggers and Scheduled Builds.
 - **Repository Triggered Quite Period.** Listens to notifications sent from the repository and builds the project

based on the changes found in the commit; used with repo triggers. This should only be used if every workflow in your project uses a repository trigger and you want AnthillPro to build every time a commit is made. See also Workflow Triggers and Scheduled Builds.

- **No Quiet Period.** Will not enforce a quiet period. While there are some instances where a quiet period is not necessary, it is best practice to always use a quiet period. If you elect not to use a quiet period, it is possible to end up with a substantial number of unnecessary builds -- many of which will be broken.
- **Should Cleanup.** If 'Yes' is selected, a cleanup will be performed prior to checking out source for a build. If 'No' is selected, no cleanup will be performed prior to source check out. With some SCM systems, it is possible to only check out changed source for a build, making the process quicker.

3. Select **agent filter type** and click **Set**.

- **Any Agent.** Selects available agent.
- **Fixed Agent.** Always selects a specific agent.
- **Scripted Agent.** Selects available agent based upon agent filter script. See Agent Filter (Selection) Scripts.

4. Click **Save**.

5. To edit an existing Agent Filter and/or Quiet Period, repeat Items One through Four.

Agent Security

User access to an agent is managed on the **Security** tab. Administrators can define what roles have access to read, write, or determine security for agents. You need administrative permissions to set agent security. See Setting Up Security.

1. Go to the **Agents > Agent** page and select the appropriate agent from the list.
2. Select the agent's **Security** tab and click **Edit**.
3. Check the **Roles** and **permissions** for this agent. See Define Roles and Using Permissions.
4. Click **Save** then **Done**.

Chapter 61. Agent Settings

Agents allow the distribution of tasks for performance and multi-platform support. To make better use of available resources and allow for higher build throughput, simultaneous builds can be performed on different agents. Agents can participate in multiple environments. This allows the use of an agent that communicates with a network deployment manager capable of deploying applications to multiple environments.

AnthillPro agents are light-weight Java processes that run on the agent machine. The agent contacts the server whenever the agent process is started. Since the agent communicates with the central server, it need not be on the same network as the central server. However, the agent must be able to open a socket connection to the server. By default, all communication between the central server and the agent is not secure. Communication may be secured using SSL. See [Configure Server-Agent SSL](#).

- **Configured Agents.** Ready to use when online.
- **Available Agents.** Installed but are not yet configured, and cannot be used until configured. See [Agent Configuration](#).
- **Ignored Agents.** Temporarily removed from service. To use an ignored agent, select it, click **Edit**, check settings, click **Set** then **Done**.

To view the Agents overview screen, go to the **Agents** page, and select the **Agent** from the menu. Agents can be organized in either ascending or descending order by selecting the Menu headings. For example, select the **Name** heading to list all agents in either alphabetical or reverse alphabetical order; or select the **Host** heading to similarly organize all agents by host name.

For organizations with a large number of agents, the agent list page provides navigated pagination. Choose to display 5, 25, 50, or 100 agents per page, and use the navigation tool bar (directly above the Operations menu) to view other pages.

The agent properties (go to **Agents > Agent > select an agent > Property type**) are also stored on the server, making searching for a property easier. When the agent starts up, it passes all of its properties to the server, which then stores it. If a property is added or changed, or the agent is upgraded, etc., the properties will be automatically updated on the server.

If an agent is not responding, it may be restarted in one of two ways:

- **Restart All Agents** will restart every online agent, even agents currently performing work. Once restart has completed, any work being performed by the agents will resume.
- **Test Communication** ensures that the server and agent are communicating properly. Once selected from the Operations menu, a dialog box will pop up describing the action. Please note that checking the communication may take a minute or longer, depending on agent load and locks.
- **Restart**, under the Operations menu, will restart an individual agent. If, at the time of restart, the agent was currently performing work, the work will resume once restart has completed.

See also [About AnthillPro Agents](#).

Using Agent Proxies

Use an agent proxy for any agents where the direct agent-server communication is prohibited. Once enabled, the

proxy allows the agent to send logs, reports and other artifacts to the server. To use an agent proxy for server-agent communication, modify the following properties in the `agent.properties` file:

- `locked/agent.http.proxy.host=`
- `locked/agent.http.proxy.port=`

Creating an Agent Proxy

Note that only previously installed agents can be set up to use a proxy. If the agent has not already been installed, see [Installing AnthillPro](#).

1. Shut down the agent.
2. With a text editor, open the `agent.properties` file located in the `\conf\agent` directory (for example, `C:\%AGENT_HOME%\conf\agent\agent.properties`).
3. Modify the `locked/agent.http.proxy.host=` property. Give the **host name** for the agent proxy. For example, `locked/agent.http.proxy.host=www.proxy.yourcompany.com`.
4. Modify the `locked/agent.http.proxy.port=` property. Give the **port** on which the agent proxy is to communicate. It should look something like `locked/agent.http.proxy.port=8900`.
5. Save changes and restart agent.

Chapter 62. Dynamically Set Agent's Environment Variables

Agent Environment Variables (not to be confused with AnthillPro Environment Properties) are set at the time of agent installation, and are stored on the central server. The agent-installation reads the environment variables of the machine the agent is being installed on and then sets the variables. Once set, AnthillPro does not check for changes in the machine's environment variables. However, you can configure the agent to dynamically set the environment variables every time the agent restarts.

When dynamically setting the variables, the agent needs two files in the `/bin` directory: one named `env.properties` and one named either `setenv.bat` (for Windows) or `setenv.sh` (for UNIX-like systems). The `setenv` shell script is responsible for dynamically generating the environment variables in a format that AnthillPro can use, as well as writing out the `env.properties` file. The `env.properties` file includes all the variables in a pattern similar to `DYNAMIC_ENV_VAR=value`.

When the agent is restarted, it will first look for the `setenv` script. If the script is found, it will be executed to update (or write out) the `env.properties` file. Then, the new variables in the `env.properties` file will be delivered to the AnthillPro server.

Prerequisites

- Access the agent's `/bin` directory.
- Ability to write a shell script that can generate the environment variables in the proper format.

Add Files to Agent's `/bin` Directory

1. **Stop** the agent.
2. **Create `setenv` file.** In a text editor, create the `setenv.bat` file if the agent is running on Windows or `setenv.sh` file if the agent is running on a UNIX-like system. Save it to the agent's `/bin` directory.

You will need to have the shell script read the environment variables and then writes them out to the `env.properties` file. The script should be similar to the example below:

```
echo DYNAMIC_ENV_VAR=value > env.properties
```

Note that the `env.properties` cannot reference other environment variables such as `PATH=%PATH%;...`, etc.

AnthillPro requires the variables to be set using this exact pattern.

3. **Restart the agent.** Upon restart, the variables will be updated. To update the environment variables in the future, simply restart the agent again.

Part XII. Tool Integrations

AnthillPro ships with a number of built-in integrations with many of the tools used throughout the project's lifecycle. The integrations, which are implemented at the job and step levels, enable AnthillPro to distribute process execution and invoke third-party tools. For example, AnthillPro can perform work such as checking out code, running a build, or firing up a virtual machine.

In addition to driving other tools, AnthillPro acts as an information hub: the integrations enable AnthillPro to exchange information with other tools, and then perform actions based on the information. For example, when AnthillPro gets the changelog from an SCM, that information is stored in the AnthillPro data warehouse and then made available through the UI. For integrations with issue trackers, AnthillPro can add a comment to an existing issue, or even create an issue in the other tool. And for testing integrations, AnthillPro collects test results and can then provide you with metrics for both a single build and for many builds over time.

AnthillPro currently supports integrations with the following tool types:

Build	Source-code Analysis	Testing
Issue Tracking	Test Coverage	Virtualization
Source Code Management		

If AnthillPro does not currently integrate with a tool you use, you can either contact us at <sales@urbancode.com> or you try writing your own integration using the Plugins feature.

Chapter 63. SCM Tools

The SCM integrations enable AnthillPro to check out code, access the changelog, and label the repository (where supported). To do this, AnthillPro is first configured with your repository type at the System level, and then each workflow is associated with the source to be build.

The SCM integrations are implemented as job steps for a build job. Once you have completed source configuration, you can use the Job Wizard to automatically add steps to your build job -- this ensures that AnthillPro will consistently checkout and build the correct code.

Each SCM integration typically performs the following for any repository:

- **Checkout.** This step enables you to define which version of code to check out from the SCM. For example, you can configure this step to checkout the latest source code; or perform a checkout based on a branch, label, date, etc. (depending on what your SCM supports).
- **Get-changelog.** The retrieved changelog is usually based on the changes made since the previous build. This step enables AnthillPro to extract data from the SCM and then store it in the AnthillPro data warehouse. Since AnthillPro stores the changelogs, it can parse the data, allowing you to override the default behavior. For example, you can select a starting point for the changelog based on criteria such as the latest production build.
- **Label.** AnthillPro can also apply a label to the source code used in the build (e.g., snapshot, baseline). This unique identifier for a build can be used to recreate a build if necessary.
- **SCM-specific commands.** For most repository types, AnthillPro can also perform tasks only supported by that particular SCM.

AccuRev

The first step in using an AccuRev repository with AnthillPro is to create a repository configuration by following the Repositories link on the System page. The configuration will allow basic information regarding AccuRev to be used by several projects. Once configured, the repository will be listed on the Repositories main page.

In AnthillPro, there are two ways to use an AccuRev repository:

- **Pooling streams.** AnthillPro creates and manages a pool of AccuRev streams to be used for the project, and does not require creation of AccuRev workspaces. Any streams included in the pool are automatically locked and should not be edited (locked streams may be viewed at **System > Lockable Resources** for those with Administrative permissions). This option is simpler than the alternative, and requires less configuration and management. See AccuRev Source Configuration with Pooling Streams.
- **Non-pooling streams.** AnthillPro does not create and maintain a pool of children streams for the project. Management of streams and workspaces must be done through AccuRev. See AccuRev Source Configuration with Non-pooling Streams.

Once the main repository has been identified, your projects can then use the AccuRev repository during workflow source configuration. During project creation, the workflow is associated with a specific AccuRev repository.

AnthillPro also provides an integration with AccuWork. The integration enables AnthillPro to utilize AccuWork's bridge between third-party issue trackers (such as Bugzilla and JIRA) and AccuRev to publish the generated issue report to AnthillPro Build Life. Once the report is published to AnthillPro, it is then be made available on the Build Life Issues tab. See AccuWork.

AccuRev Prerequisites

- AccuRev must already be installed. See AccuRev documentation [<http://www.accurev.com/documentation.html>].
- You must have administrative permissions. See *Manage Security*.
- The AccuRev command path must be available to complete the configuration.
- *AccuRev Login*. For versions 4.5.1 and above, the new AccuRev authentication scheme now requires a manual login (usually daily) in order for an AnthillPro agent to access the repository. To override this behavior, you can automate login by using the `-n` option with the AccuRev login command. This will cache the credentials so that AccuRev no longer asks for a password for that account. If you want to automate login for the AnthillPro agent(s):
 - Login to AccuRev with the account that the AnthillPro agent uses to access the repository.
 - Run the AccuRev login command with the `-n` flag: `accurev login -n`.
 - If you have AnthillPro agents that login to AccuRev as different users, you will need to repeat this operation for those agents as well.

Set Up AccuRev Repository

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **AccuRev** from the drop-down menu and click **Set**.
4. Configure repository:
 - **Name** the repository. This is the name AnthillPro will use to identify this repository. This name does not correspond to the actual repository, and is simply an identifying label used by the AnthillPro system.
 - **Description**. Give an optional description.
 - **Command Path**. Provide the location of the AccuRev executable if it is not in the system path.
 - **Pool Streams**. If not using an AnthillPro pool of streams, do not check the box and proceed to Item 5 (see *AccuRev Source Configuration with Non-pooling Streams*).

To use an AnthillPro-created pool of AccuRev streams across projects using this repository, check the box (see *AccuRev Source Configuration with Pooling Streams*).
 - **Max Number in Pool**. Give the maximum number of AccuRev streams AnthillPro will create in the pool (if using the Pool Streams option).
5. Click **Set**.
6. Select the **Security** tab and click the **Edit** button. Determine **permissions**, click **Save**, and then click **Done**.
7. Click **Done**.

AccuRev Source Configuration with Pooling Streams

Once the main AccuRev repository is identified (see Set Up AccuRev Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the AccuRev repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Configure, or Edit Source, link under the Source Config menu on the Workflow Main page.)

1. Once AccuRev is set up with AnthillPro (see Set Up AccuRev Repository), create a project.
2. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see Create a New Workflow before continuing. Configure the source on the workflow:
 - **Repository.** Select the appropriate AccuRev repository from the drop-down menu. All AccuRev repositories configured with AnthillPro (see Set Up AccuRev Repository) will appear in the drop-down menu. Make sure to select the correct repository for this project.

- **Working Directory Script.** Select the Working Directory Script from the drop-down menu. The working directory is the location where the agent is going to run its commands. For example, if the working directory is `C:\projects\Project_A\` every command AnthillPro issues will be from that specific directory.

If no Working Directory Script is configured in the job, the script selected here will be used. See Working Directory Scripts.

- **Directory Offset.** Give the relative directory path from the working directory where this stream should be pulled. Leave blank for the root of the working directory.

This option is especially useful when using multiple Sources (see Add Additional Source with Pooling Streams).

- **Stream Name.** Provide the name of the parent stream to back the pooled stream with. This field accepts small scripts for including properties, etc.
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/ -`
- for any number of any characters except \ and /: `* -`
- for any single character / including \ and / Changelog file paths containing drive letter or any prefixes: `? -`

Make sure to include the * at the beginning of the path.

Example exclude paths: `-**/abc/**/*` Except for: `+**/abc/*.html`. This will exclude everything under the abc directory except for the HTML files in it.

3. Click **Save**.

Add Additional AccuRev Source with Pooling Streams

If adding additional sources to the workflow is necessary, set up is similar to that of the initial source configuration (see AccuRev Source Configuration with Pooling Streams). When adding additional sources, it is advisable (though not required) to create a new directory offset for each additional source.

Once an additional source has been added to the workflow, it will appear on the workflow Main tab under the Source Config menu, and may be edited or deleted using the icons under the Actions menu.

To add an additional source to an existing workflow:

1. Go to **Administration** and select the workflow that an additional source is to be added to.
2. On the workflow **Main** page, select the **Add Additional Source** link under the Source Config menu.
3. Configure source:
 - **Directory Offset.** Give the relative directory path from the working directory where this stream should be pulled. Leave blank for the root of the working directory. For example, enter *Test* to create the offset named "Test".

This option is especially useful when using multiple Sources. Each source may use a different offset, making it easier to pull different streams down at the same time.
 - **Stream Name.** Provide the name of the parent stream to back the pooled stream with. This field accepts small scripts for including properties, etc.
4. Click **Save**.

AccuRev Source Configuration with Non-pooling Streams

Once the main AccuRev repository is identified (see Set Up AccuRev Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the AccuRev repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Configure, or Edit Source, link under the Source Config menu on the Workflow Main page.)

1. Once AccuRev is set up with AnthillPro (see Set Up AccuRev Repository), create a project.
2. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see Create a New Workflow before continuing. Configure the source on the workflow:
 - **Repository.** Select the appropriate AccuRev repository from the drop-down menu. All AccuRev repositories configured with AnthillPro (see Set Up AccuRev Repository) will appear in the drop-down menu. Make sure to select the correct repository for this project.
 - **Working Directory Script.** Select the Working Directory Script from the drop-down menu. The working directory is the location where the agent is going to run its commands. For example, if the working directory is `C:\projects\Project_A\` every command AnthillPro issues will be from that specific directory.

If no Working Directory Script is configured in the job, the script selected here will be used. See Working Directory Scripts.

- **Directory Offset.** Give the relative directory path from the working directory where this stream should be pulled. Leave blank for the root of the working directory. For example, enter *Test* to create the offset named "Test".

This option is especially useful when using multiple sources. Each source may use a different offset, making it easier to pull different streams down at the same time. See Add Additional Source with Non-pooling Streams.

- **Stream Name.** Provide the name of the stream to build from. This field accepts small scripts for including properties, etc. *This may not be a pass-through stream. Only use a child stream off of the development stream where the changes are committed.*
- **Workspace Name.** Give the name of the AccuRev workspace to use. This field accepts small scripts for including properties, etc. *Do not include the _user-name at the end of the workspace name, this will be determined by the credentials in the AccuRev Repository Configuration.*
- **History Stream Name.** Give the name of the AccuRev parent stream from which the history of changes will be collected. *Use the Stream Name option only if this stream is different from the stream name listed above (see Stream Name).* This option is helpful when labeling a history stream, to ensure that the proper snapshot is used for labeling. See Label On History Stream below.
- **Disable Time Locking.** Check the box to override the automatic time lock that AnthillPro sets on the backing stream. Note that without the time lock, there is no guarantee that a build can be reproduced.
- **Label On History Stream.** Check the box create a snapshot of the parent/history stream. If a child stream as used as the backing stream, any labelling of that child stream may produce inconsistent results. The only way to ensure proper labeling results is to create a snapshot on the parent/history stream.

When using this option, make sure that the AccuRev parent stream is defined in the History Stream Name. See History Stream Name above.

- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: **/ -
- for any number of any characters except \ and /: * -
- for any single character / including \ and / Changelog file paths containing drive letter or any prefixes: ? -

Make sure to include the * at the beginning of the path.

*Example exclude paths: -**/abc/**/* Except for: +**/abc/*.html. This will exclude everything under the abc directory except for the HTML files in it.*

3. Click **Save**.

Add Additional AccuRev Source with Non-pooling Streams

If adding additional sources to the workflow is necessary, set up is similar to that of the initial source configuration (see AccuRev Source Configuration with Non-pooling Streams). When adding additional sources, it is advisable (though not required) to create a new directory offset for each additional source.

Once an additional source has been added to the workflow, it will appear on the workflow Main tab under the Source Config menu, and may be edited or deleted using the icons under the Actions menu.

To add an additional source to an existing workflow:

1. Go to **Administration** and select the workflow that an additional source is to be added to.

2. On the workflow **Main** page, select the **Add Additional Source** link under the Source Config menu.
3. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see [Create a New Workflow](#) before continuing. Configure the source on the workflow:

- **Directory Offset.** Give the relative directory path from the working directory where this stream should be pulled. Leave blank for the root of the working directory. For example, enter *Test* to create the offset named "Test".

This option is especially useful when using multiple sources. Each source may use a different offset, making it easier to pull different streams down at the same time.

- **Stream Name.** Provide the name of the stream to build from. This field accepts small scripts for including properties, etc. *This may not be a pass-through stream. Only use a child stream off of the development stream where the changes are committed.*
- **Workspace Name.** Give the name of the AccuRev workspace to use. This field accepts small scripts for including properties, etc. *Do not include the `_user-name` at the end of the workspace name*, this will be determined by the credentials in the AccuRev Repository Configuration.
- **History Stream Name.** Give the name of the AccuRev parent stream from which the history of changes will be collected. *Use the Stream Name option only if this stream is different from the stream name listed above (see Stream Name).* This option is helpful when labeling a history stream, to ensure that the proper snapshot is used for labeling. See [Label On History Stream](#) below.
- **Disable Time Locking.** Check the box to override the automatic time lock that AnthillPro sets on the backing stream. Note that without the time lock, there is no guarantee that a build can be reproduced.
- **Label On History Stream.** Check the box create a snapshot of the parent/history stream. If a child stream as used as the backing stream, any labelling of that child stream may produce inconsistent results. The only way to ensure proper labeling results is to create a snapshot on the parent/history stream.

When using this option, make sure that the AccuRev parent stream is defined in the History Stream Name. See [History Stream Name](#) above.

4. Click **Save**.

ClearCase

AnthillPro provides a number of integrations with ClearCase, each based on different ClearCase usage scenarios. Which integration you use will depend on your particular needs and your organizational/team ClearCase set up. Generally, the integrations fall into one of two categories:

- **ClearCase Base.** AnthillPro supports both dynamic and snapshot views. In addition to the legacy integrations (ClearCase Base Dynamic View and ClearCase Base Snapshot View), AnthillPro also offers newer plugin integrations (see [ClearCase Base Dynamic \[Plugin\]](#) and [ClearCase Base Snapshot \[Plugin\]](#)). While the plugin versions are similar to the legacy integrations, they do offer additional functionality. Notably, the ability to add multiple source configurations to an originating (build) workflow and additional configuration options.
- **ClearCase UCM.** If you use UCM, you can configure the ClearCase UCM Dynamic View and/or the ClearCase UCM Snapshot View integration. It's worth noting that a single originating (build) workflow can have only one view-type set up during source configuration. If you use multiple views within the same project, you will need to configure a separate workflow for each view. UCM versions of ClearCase are not supported by the plugin integrations.

In addition, AnthillPro also provides an integration with ClearQuest that can be used in conjunction with any of the ClearCase integrations.

ClearCase Base Dynamic (Plugin)

The first step in using a ClearCase Base Dynamic Plugin repository with AnthillPro is to configure it by following the Repositories link on the System page. The configuration will allow basic information regarding ClearCase to be reused by several project workflows. Once configured, the repository will be listed on the Repositories main page.

Once the main repository has been identified, your projects can then use the ClearCase repository. During project creation, associate the workflow with a specific ClearCase repository and specify the Load Rules. See ClearCase Base Dynamic (Plugin) Source Configuration.

See also ClearQuest.

ClearCase Base Dynamic (Plugin) Prerequisites

- ClearCase must already be installed. See ClearCase documentation [<http://www-01.ibm.com/support/docview.wss?rs=984&uid=swg21239261>].
- You must have administrative permissions. See Manage Security.
- The cleartool executable and the Load Rules must be available to complete the configuration.
- If you are not familiar with ClearCase, it is recommended to consult with your ClearCase administrator while configuring the integration.

Set Up ClearCase Base Dynamic (Plugin) Repository

The integration is available as an AnthillPro plugin for version 3.7 and above. For some AnthillPro 3.7 versions, you will need to download the integration (called **ClearCase Base Dynamic Plugin**) from Supportal [<https://support.urbancode.com/tasks/login/LoginTasks/login>] and then upload it to your server. Once uploaded, ensure the Plugin is active. From there, configure the repository. The information you give here is system wide, and will be used by your ClearCase projects. If you have multiple ClearCase repositories, you will need to configure the integration for each one ... ensuring each has a unique name.

1. If not already done so, ensure the **ClearCase Base Dynamic Plugin** has been uploaded to the server. See Using AnthillPro Plugins.
2. Go to **System > Repositories** from the **Project Support** menu.
3. On the **Repositories** page, click the **Create New** button.
4. Select **ClearCase Base Dynamic** from the drop-down menu and click **Set**.
5. Configure repository:
 - **Name** the repository. This is the name AnthillPro will use to identify this repository. This name does not correspond to the actual repository, and is simply an identifying label used by the AnthillPro system. If you are configuring multiple repositories, ensure the name you give is unique. This name will be used as part of source configuration.
 - **Description**. Give an optional description.
 - **Command Path**. Provide the location of the ClearCase executable if it is not in the system path on the ma-

chine that will run the build.

6. Click **Set** then **Done**.

ClearCase Base Dynamic (Plugin) Source Configuration

Once the main ClearCase repository is identified (see Set Up ClearCase Base Dynamic (Plugin) Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the ClearCase repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Configure, or Edit Source, link under the Source Config menu on the Workflow Main page.)

In addition, the plugin integration allows you to add multiple source configurations to the originating (build) workflow. It's worth noting that this feature also allows you to configure multiple views (Snapshot and Dynamic) for a build workflow as well as different project locations. How, when and why you use this feature depends on your individual processes. Generally, you will only need one source configuration per build workflow.

1. Once ClearCase is set up with AnthillPro (see Set Up ClearCase Base Dynamic (Plugin) Repository), create a project.
2. When prompted, select the integration you configured in the previous section. If necessary, you can exit the configuration process and return to it later.
3. Go to the originating workflow that was created as part of the project creation process (if you chose not to continue in the previous step) and configure the source on the workflow:
 - **Working Directory Script.** Select the Working Directory to be used for this source configuration. The working directory is where the checked-out source will be placed in and where the build occurs. Under most circumstances, the **Default Project Working Directory** should suffice. For more Working Directory Scripts, see the Dev Kit (via the **tools** link in the upper right-hand corner of the UI).
 - **Name.** Give a name for this source configuration. The name given here is used to identify the configuration. This can come in handy if you plan on adding multiple configurations to the same workflow.
 - **View Storage Directory.** Give the location of the view on the local machine. This is the volume that views are mounted on (e.g., M:\) On Windows this will usually be of the form //host/share/view_storage.
 - **Load Rules.** Give the Load Rules configured in the View Configuration Specifications to tell AnthillPro what projects, branches, and labels to use. AnthillPro requires the specs to be in the form `<vob-name><path>:<branch>:<label>`. Each Load Rule must be input on a separate line.

The Load Rules also support AnthillPro properties. E.g., `${property:vob.tag}:/int_dc_10.02`.

If the View Load Rules are configured with the value `BaseModeTest:Anthill-Example::`, then they are input as `vob-name BaseModeTest` (following the form `<vob-name><path>:<branch>:<label>`) in the Load Rules field. Note that the Source Config deals with the `vob-name BaseModeTest` and that the project is located in the `Anthill-Example` directory under our VOB.

To build from the branch named `test`, the Load Rule line would be `BaseModeTest:Anthill-Example:test:` (all builds performed on the test branch).

To perform a build on label `1.0.3` the Load Rule would be `BaseModeTest:Anthill-Example::1.0.3`.

To build from label 1.0.3 of branch test the Load Rule would be BaseMode-Test:Anthill-Example:test:1.0.3.

Example Load Rule Configuration

For the load rules below, AnthillPro needs to know the path and optional branch and label the configspec is checking out in order to find the changes and then apply any needed labels. For example, the branch and label would come from lines starting with element, paths come from the lines starting with load, etc.

```
element* CHECKEDOUT

element/elementName/... ../elementNameApplication/LATEST
element/elementName/... ../iter2_elementNameApplication/LATEST -mkbranch elementNameApp
element/mmisPortal/... /main/LATEST -mkbranch iter2_portal
element* ../element_customization/LATEST
element* ../main/LATEST
load /elementNamePortal
load /elementNameCommon
load /elementNameFramework
```

For example, if you look at the first line in the load rules:

```
element/elementName/... ../elementNameApplication/LATEST
```

means that for path /elementName you need to get the code from the elementNameApplication branch (LATEST is the default label so you can ignore it when configuring AnthillPro). This translates in the following load rule (using the pattern VOB:/path_in_vob:branch:label):

```
elementName:/:elementNameApplication:
```

In the same manner, the second line translates into

```
elementName:/:iter2_elementNameApplication:
```

However, the -mkbranch in the line is ignored when configuring AnthillPro since -mkbranch says it's not about getting code but committing code back.

For the line element* ../element_customization/LATEST means that for every load rule we need to get the code from the element_customization branch. This requires one load rule be configured in AnthillPro per load rule in ClearCase. Thus, you would have something like:

```
elementName:/:element_customization:
```

```
elementNameCommon:/:element_customization:
```

```
elementNameFramework:/:element_customization:
```

Note that element* ../main/LATEST also needs to be addressed. In the example, you will need to get code from the main branch. This should look something like:

```
elementNamePortal:/::
```

```
elementNameCommon:/::
```

```
elementNameFramework:/::
```

- **VOB Tag Root.** Only use this setting if you are running cross-platform builds including Windows and UNIX and using multi-component vobs. When specifying the vob tag root do not use any starting or trailing slashes. The value is usually vobs.
- **Create global labels.** Check the box to create global labels. You can also specify in which VOBs those labels will be created (see below). If you do not specify any VOBs (below), the Load Rules will be used to determine all the VOBs that are used, and labels will be created in those VOBs.
- **Create label in VOBs.** Please list the names of the VOBs (one per line) where the global labels should be created. If you are not creating global labels (option above is not selected) you can list all the VOBs used in your config spec or leave the field empty and the Load Rules will be used to determine where the labels need to be created.
- **Delete view-private files during cleanup.** Select here if you would like AnthillPro to detect and delete any view-private files in the local view. This is highly recommended unless your build script handles cleanup of the view. Only the paths specified in the Load Rules above will be checked for view-private files.
- **No-checkout.** Check the box to pass the `-ncO` flag to the `quietperiod lshistory` call. Typically, this option is used when you do not want non-checkouts in the history to trigger builds.
- **Lock Objects.** Please list the names of the objects (one per line) that you would like locked/unlocked during the build process. Use the following format: `type:Name@VOB` (or refer to the `lock/unlock ClearCase` command reference for more details). *Be advised that if you leave this field empty, there is nothing to prevent an update of the dynamic view while there is a build in progress.*
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File Filters.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/ -`
- for any number of any characters except \ and /: `* -`
- for any single character / including \ and / Changelog file paths containing drive letter or any prefixes: `? -`

Make sure to include the `*` at the beginning of the path.

Example exclude paths: `-**/abc/**/*` Except for: `+**/abc/*.html`. This will exclude everything under the abc directory except for the HTML files in it.

- **Repository.** Select the repository configured in the previous section. If more than one configuration is present, ensure the correct repository configuration is chosen.

4. Click **Save**.

ClearCase Base Dynamic View

The first step in using a ClearCase Base Dynamic View repository with AnthillPro is to configure it by following the Repositories link on the System page. The configuration will allow basic information regarding ClearCase to be reused by several project workflows. Once configured, the repository will be listed on the Repositories main page.

Once the main repository has been identified, your projects can then use the ClearCase repository. During project

creation, associate the workflow with a specific ClearCase repository and determine the mode, view, and specify the Load Rules. See ClearCase Base Dynamic View Source Configuration.

See also ClearQuest.

ClearCase Base Dynamic View Prerequisites

- ClearCase must already be installed. See ClearCase documentation [<http://www-01.ibm.com/support/docview.wss?rs=984&uid=swg21239261>].
- You must have administrative permissions. See Manage Security.
- The cleartool executable and the Load Rules must be available to complete the configuration.
- If you are not familiar with ClearCase, it is recommended to consult with your ClearCase administrator while configuring the integration.

Set Up ClearCase Base Dynamic View Repository

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **ClearCase** from the drop-down menu and click **Set**.
4. Configure repository:
 - **Name** the repository. This is the name AnthillPro will use to identify this repository. This name does not correspond to the actual repository, and is simply an identifying label used by the AnthillPro system.
 - **Description**. Give an optional description.
 - **Command Path**. Provide the location of the ClearCase executable if it is not in the system path.
5. Click **Set**.
6. Select the **Security** tab and click the **Edit** button. Determine **permissions**, click **Save**, and then click **Done**.
7. Click **Done**.

ClearCase Base Dynamic View Source Configuration

Once the main ClearCase repository is identified (see Set Up ClearCase Base Dynamic View Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the ClearCase repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Configure, or Edit Source, link under the Source Config menu on the Workflow Main page.)

1. Once ClearCase is set up with AnthillPro (see Set Up ClearCase Base Dynamic View Repository), create a project.
2. Select the Mode and View Type once ClearCase has been selected as the repository type.

- **Mode.** Choose **Base**.
 - **View Type.** Choose **Use Dynamic**.
 - Click **Select**.
3. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see [Create a New Workflow](#) before continuing. Configure the source on the workflow:
- **View Storage Directory.** Give the location of the view on the local machine. On Windows this will be of the form `//host/share/view_storage`.
 - **Load Rules.** Give the Load Rules configured in the View Configuration Specifications to tell AnthillPro what projects, branches, and labels to use. AnthillPro requires the specs to be in the form `<vob-name><path>:<branch>:<label>`. Each Load Rule must be input on a separate line.

The Load Rules also support AnthillPro properties. E.g., `${property:vob.tag}:/int_dc_10.02`.

If the View Load Rules are configured with the value `BaseModeTest:Anthill-Example::`, then they are input as `vob-name BaseModeTest` (following the form `<vob-name><path>:<branch>:<label>`) in the Load Rules field. Note that the Source Config deals with the `vob-name BaseModeTest` and that the project is located in the `Anthill-Example` directory under our VOB.

To build from the branch named `test`, the Load Rule line would be `BaseModeTest:Anthill-Example:test:` (all builds performed on the test branch).

To perform a build on label `1.0.3` the Load Rule would be `BaseModeTest:Anthill-Example::1.0.3`.

To build from label `1.0.3` of branch `test` the Load Rule would be `BaseModeTest:Anthill-Example:test:1.0.3`.

Example Load Rule Configuration

For the load rules below, AnthillPro needs to know the path and optional branch and label the configspec is checking out in order to find the changes and then apply any needed labels. For example, the branch and label would come from lines starting with `element`, paths come from the lines starting with `load`, etc.

```
element* CHECKEDOUT
element/elementName/... ../elementNameApplication/LATEST
element/elementName/... ../iter2_elementNameApplication/LATEST -mkbranch elementNameApp
element/mmisPortal/... /main/LATEST -mkbranch iter2_portal
element* ../element_customization/LATEST
element* ../main/LATEST
load /elementNamePortal
load /elementNameCommon
load /elementNameFramework
```

For example, if you look at the first line in the load rules:

```
element/elementName/... ../elementNameApplication/LATEST
```

means that for path `/elementName` you need to get the code from the `elementNameApplication` branch

LATEST is the default label so you can ignore it when configuring AnthillPro). This translates in the following load rule (using the pattern VOB: /path_in_vob:branch:label):

```
elementName://elementNameApplication:
```

In the same manner, the second line translates into

```
elementName://iter2_elementNameApplication:
```

However, the `-mkbranch` in the line is ignored when configuring AnthillPro since `-mkbranch` says it's not about getting code but committing code back.

For the line `element* .../element_customization/LATEST` means that for every load rule we need to get the code from the `element_customization` branch. This requires one load rule be configured in AnthillPro per load rule in ClearCase. Thus, you would have something like:

```
elementName://element_customization:
```

```
elementNameCommon://element_customization:
```

```
elementNameFramework://element_customization:
```

Note that `element* .../main/LATEST` also needs to be addressed. In the example, you will need to get code from the main branch. This should look something like:

```
elementNamePortal://:
```

```
elementNameCommon://:
```

```
elementNameFramework://:
```

- **Create global labels.** Check the box to create global labels. You can also specify in which VOBs those labels (see below). If you do not specify any VOBs (below), the Load Rules will be used to determine all the VOBs that are used, and labels will be created in those VOBs.
- **Create labels in VOBs.** Please list the names of the VOBs (one per line) where the global labels should be created. If you are not creating global labels (option above is not selected) you can list all the VOBs used in your config spec or leave the field empty and the Load Rules will be used to determine where the labels need to be created.
- **Delete view-private files during cleanup.** Select here if you would like AnthillPro to detect and delete any view-private files in the local view. This is highly recommended unless your build script handles cleanup of the view. Only the paths specified in the Load Rules above will be checked for view-private files.
- **Lock Objects.** Please list the names of the objects (one per line) that you would like locked/unlocked during the build process. Use the following format: `type:Name@VOB` (or refer to the lock/unlock ClearCase command reference for more details). *Be advised that if you leave this field empty, there is nothing to prevent an update of the dynamic view while there is a build in progress.*
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/` -
- for any number of any characters except `\` and `/`: `*` -
- for any single character / including `\` and `/` / Changelog file paths containing drive letter or any prefixes: `?` -

Make sure to include the `*` at the beginning of the path.

Example exclude paths: `-**/abc/**/*` Except for: `+**/abc/*.html`. This will exclude everything under the abc directory except for the HTML files in it.

4. Click **Save**.

ClearCase Base Snapshot (Plugin)

The first step in using a ClearCase Base Snapshot Plugin repository with AnthillPro is to configure it by following the Repositories link on the System page. The configuration will allow basic information regarding ClearCase to be reused by several project workflows. Once configured, the repository will be listed on the Repositories main page.

Once the main repository has been identified, your projects can then use the ClearCase repository. During project creation, associate the workflow with a specific ClearCase repository and specify the ChangeLog Rules. See ClearCase Base Snapshot (Plugin) Source Configuration.

During source configuration, you must select a View Strategy. The strategy will determine how AnthillPro interacts with existing views and determine when AnthillPro should create a new view. Select a strategy that either (a.) creates a new view with every build; (b.) create a new view if and only if a view does not exist; or (c.) does not create a new view and uses an existing view. Only one view may be configured on an individual workflow; however, if a project has multiple workflows, a separate View Strategy may be used for each workflow.

See also ClearQuest.

ClearCase Base Snapshot (Plugin) Prerequisites

- ClearCase must already be installed. See ClearCase documentation [<http://www-01.ibm.com/support/docview.wss?rs=984&uid=swg21239261>].
- You must have administrative permissions. See Manage Security.
- The cleartool executable and the ChangeLog Rules must be available to complete the configuration.
- If you are not familiar with ClearCase, it is recommended to consult with your ClearCase administrator while configuring the integration.

Set Up ClearCase Base Snapshot (Plugin) Repository

The integration is available as an AnthillPro plugin for version 3.7 and above. For some AnthillPro 3.7 versions, you will need to download the integration (called **ClearCase Base Snapshot Plugin**) from Supportal [<https://support.urbancode.com/tasks/login/LoginTasks/login>] and then upload it to your server. Once uploaded, ensure the Plugin is active. From there, configure the repository. The information you give here is system wide, and will be used by your ClearCase projects. If you have multiple ClearCase repositories, you will need to configure the integration for each one ... ensuring each has a unique name.

1. If not already done so, ensure the **ClearCase Base Snapshot Plugin** has been uploaded to the server. See Using AnthillPro Plugins.
2. Go to **System > Repositories** from the **Project Support** menu.
3. On the **Repositories** page, click the **Create New** button.
4. Select **ClearCase Base Snapshot** from the drop-down menu and click **Set**.
5. Configure repository:
 - **Name** the repository. This is the name AnthillPro will use to identify this repository. This name does not correspond to the actual repository, and is simply an identifying label used by the AnthillPro system. If you are configuring multiple repositories, ensure the name you give is unique. This name will be used as part of source configuration.
 - **Description**. Give an optional description.
 - **Command Path**. Provide the location of the ClearCase executable if it is not in the system path on the machine that will run the build.
6. Click **Set** then **Done**.

ClearCase Base Snapshot (Plugin) Source Configuration

Once the main ClearCase repository is identified (see Set Up ClearCase Base Snapshot (Plugin) Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the ClearCase repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Configure, or Edit Source, link under the Source Config menu on the Workflow Main page.)

During configuration, you will need to ensure that the appropriate fields are configured for the view strategy you select. Any fields identified as "required for all views" or "optional for all views" are used regardless of the view strategy you use. Any field that is view-specific is identified as such.

In addition, the plugin integration allows you to add multiple source configurations to the originating (build) workflow. It's worth noting that this feature also allows you to configure multiple views (Snapshot and Dynamic) for a build workflow as well as different project locations. How, when and why you use this feature depends on your individual processes. Generally, you will only need one source configuration per build workflow.

1. Once ClearCase is set up with AnthillPro (see Set Up ClearCase Base Snapshot (Plugin) Repository), create a project.
2. When prompted, select the integration you configured in the previous section. If necessary, you can exit the configuration process and return to it later.
3. Go to the originating workflow that was created as part of the project creation process (if you chose not to continue in the previous step) and configure the source on the workflow:
 - **Name (required for all views)**. Give a name for this source configuration. The name given here is used to identify the configuration. This can come in handy if you plan on adding multiple configurations to the same workflow.
 - **ChangeLog Rules (required for all views)**. Give the ChangeLog Rules configured in the View Configuration Specifications to tell AnthillPro what projects, branches, and labels to use. AnthillPro requires the specs to be in the form `<vob-name><path> : <branch> : <label>`. Each Rule must be input on a separate line.

The Rules also support AnthillPro properties. E.g., `${property:vob.tag}:/int_dc_10.02`.

If the Rules are configured with the value `BaseModeTest:Anthill-Example::`, then they are input as `vob-name BaseModeTest` (following the form `<vob-name><path>:<branch>:<label>`) in the Rules field. Note that the Source Config deals with the `vob-name BaseModeTest` and that the project is located in the `Anthill-Example` directory under our VOB.

To build from the branch named `test`, the Rule would be `BaseModeTest:Anthill-Example:test:` (all builds performed on the test branch).

To perform a build on label `1.0.3` the Rule would be `BaseModeTest:Anthill-Example::1.0.3`.

To build from label `1.0.3` of the branch named `test`, the Rule would be `BaseModeTest:Anthill-Example:test:1.0.3`.

Example ChangeLog Rule Configuration

For the Rules below, AnthillPro needs to know the path and optional branch and label the configspec is checking out in order to find the changes and then apply any needed labels. For example, the branch and label would come from lines starting with `element`, paths come from the lines starting with `load`, etc.

```
element* CHECKEDOUT

element/elementName/... ../elementNameApplication/LATEST
element/elementName/... ../iter2_elementNameApplication/LATEST -mkbranch elementNameApp
element/mmisPortal/... /main/LATEST -mkbranch iter2_portal
element* ../element_customization/LATEST
element* ../main/LATEST
load /elementNamePortal
load /elementNameCommon
load /elementNameFramework
```

For example, if you look at the first line in the Rules:

```
element/elementName/... ../elementNameApplication/LATEST
```

means that for path `/elementName` you need to get the code from the `elementNameApplication` branch (LATEST is the default label so you can ignore it when configuring AnthillPro). This translates in the following Rule (using the pattern `VOB:/path_in_vob:branch:label`):

```
elementName:/:elementNameApplication:
```

In the same manner, the second line translates into

```
elementName:/:iter2_elementNameApplication:
```

However, the `-mkbranch` in the line is ignored when configuring AnthillPro since `-mkbranch` says it's not about getting code but committing code back.

For the line `element* ../element_customization/LATEST` means that for every Rule we need to get the code from the `element_customization` branch. This requires one Rule be configured in AnthillPro per Rule in ClearCase. Thus, you would have something like:

```
elementName:/:element_customization:
```

```
elementNameCommon:/:element_customization:
```

```
elementNameFramework:/:element_customization:
```

Note that `element* .../main/LATEST` also needs to be addressed. In the example, you will need to get code from the main branch. This should look something like:

```
elementNamePortal:/::
```

```
elementNameCommon:/::
```

```
elementNameFramework:/::
```

- **VOB Tag Root (optional for all views).** Only use this setting if you are running cross-platform builds including Windows and UNIX and using multi-component vobs. When specifying the vob tag root do not use any starting or trailing slashes. The value is usually vobs.
- **Create global labels (optional for all views).** Check the box to create global labels. You can also specify in which VOBs those labels will be created (see below). If you do not specify any VOBs (below), the Load Rules will be used to determine all the VOBs that are used, and the labels will be created in those VOBs.
- **Create labels in VOBs (optional for all views).** Please list the names of the VOBs (one per line) where the global labels should be created. If you are not creating global labels (option above is not selected) you can list all the VOBs used in your config spec or leave the field empty, and the Load Rules will be used to determine where the labels need to be created.
- **Delete view-private files during cleanup (optional for Already Exists and Doesn't Exist views).** Select here if you would like AnthillPro to detect and delete any view-private files in the local view. This is highly recommended unless your build script handles cleanup of the view. Only the paths specified in the Rules above will be checked for view-private files.
- **Host name (optional for Everytime and Doesn't Exist views).** Provides the value for the `-host` parameter to be used in the `mkview` command. Use this field only if the view and storage are in different locations.
- **Global storage path [`gpath`] (optional for Everytime and Doesn't Exist views).** Provides the value for the `-gpath` parameter to be used in the `mkview` command. Use this field only if the view and storage are in different locations.
- **Host storage path [`hpath`] (optional for Everytime and Doesn't Exist views).** Provides the value for the `-hpath` parameter to be used in the `mkview` command. Use this field only if the view and storage are in different locations.
- **View Strategy.** Select a view strategy. Which fields you configure depends on the view you select:
 - *Everytime.* Create a new view every time we build. If this option is chosen, AnthillPro will check for an existing snapshot view and remove it before creating a new view with every build. This option will ensure that the view always represents the state of the current build.
 - *Doesn't Exist.* Create a new view only if one does not exist already. AnthillPro will check for an existing snapshot view. If one is found, a new view will not be created. If no existing view is found, AnthillPro will create a new one. This option allows you to use an existing view; however, that view may not represent the state of the latest build.
 - *Already Exists.* The view already exists and will be used every time. AnthillPro will not create a new view and use the existing view. This option will always use the view that previously exists. If no view exists, one

must be created prior to using this option.

- **View Location (required for all views).** If strategy is Already Exists, the location of the view on the local machine. Otherwise, The location of the view storage directory on the ClearCase server where the views are stored. For Windows operating systems this should always be a UNC path of the form /host/share/view_location.
- **View Name (required for all views).** Give the name to use for the view when creating and deleting. Ensure that the name given here is unique. Note the agent that will be running the job also needs to have a unique view name. To accomplish this, scripts can be used. For example, using the following script will ensure the name is unique (you can copy the script into the text box as is):

```
{bsh:ProjectLookup.getCurrent().getName()}_view_{bsh:AgentHelper.getCurrent().getName()}
```

- **Config Spec (optional for Everytime and Doesn't Exist views).** Provide the config spec to be used for the new view. Please replace any labels used in the config spec with a token.

For example: /main/LATEST should become /main/\${LATEST} and /main/test/1.0.3 should become /main/test/\${1.0.3}.

Place a time token after any element rules that do not load from a particular label: element * /main/LATEST should become element * /main/LATEST \${time.token} and element * /main/test/LATEST should become element * /main/test/LATEST \${time.token}.

Do not assign the element * /main/test/1.0.3 a time token.

- **Check New Config Spec (optional for Everytime and Doesn't Exist views).** Select here if you want to compare the newly set configspec against the user-specified configspec. This option allows you to determine success based on the new configspec set on the view rather than the outcome of the setcs command.
- **Label to check out from (optional for Everytime and Doesn't Exist views).** Give the label to check out source from (if any). Leave blank if not checking out via a label.
- **Tmode (optional for Everytime and Doesn't Exist views).** Select the correct text mode to use -- the standard ClearCase options are available. See About text modes for view [http://publib.boulder.ibm.com/infocenter/cchelp/v7r0m0/index.jsp?topic=/com.ibm.rational.clearcase.hlp.doc/cc_main/about_view_text_mode.htm] at the IBM Software information center for more.
- **Use VOB Time (optional for Everytime and Doesn't Exist views).** Check to box if you want to use the time stamp from when the file was checked in. If you leave this blank, the time the file was checked out will be used.
- **Use Tags (optional for all views).** Select here if you want your view to be globally created. Having this option checked can cause problems with creating and dropping snapshots if you wish to have the same snapshot on multiple agents.
- **No-checkout (optional for all views).** Check the box to pass the -nco flag to the quietperiod lshistory call. Typically, this option is used when you do not want non-checkouts in the history to trigger builds.
- **Users to exclude from changelog (optional for all views).** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File Filters (optional for all views).** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: **/ -
- for any number of any characters except \ and /: * -
- for any single character / including \ and / Changelog file paths containing drive letter or any prefixes: ? -

Make sure to include the * at the beginning of the path.

Example exclude paths: -**/abc/**/* Except for: +**/abc/*.html. This will exclude everything under the abc directory except for the HTML files in it.

- **Repository (required for all views).** Select the repository configured in the previous section. If more than one configuration is present, ensure the correct repository configuration is chosen.

4. Click **Save**.

ClearCase Base Snapshot View

The first step in using a ClearCase Base Snapshot View repository with AnthillPro is to configure it by following the Repositories link on the System page. The configuration will allow basic information regarding ClearCase to be reused by several project workflows. Once configured, the repository will be listed on the Repositories main page.

Once the main repository has been identified, your projects can then use the ClearCase repository. During project creation, associate the workflow with a specific ClearCase repository and determine the mode, view, and specify the Load Rules. See ClearCase Base Snapshot View Source Configuration.

During source configuration, you must select a View Strategy. The strategy will determine how AnthillPro interacts with existing views and determine when AnthillPro should create a new view. Select a strategy that either (a.) creates a new view with every build; (b.) create a new view if and only if a view does not exist; or (c.) does not create a new view and uses an existing view. Only one view may be configured on an individual workflow; however, if a project has multiple workflows, a separate View Strategy may be used for each workflow. See View Strategy.

See also ClearQuest.

ClearCase Base Snapshot View Prerequisites

- ClearCase must already be installed. See ClearCase documentation [<http://www-01.ibm.com/support/docview.wss?rs=984&uid=swg21239261>].
- You must have administrative permissions. See Manage Security.
- The cleartool executable and the Load Rules must be available to complete the configuration.
- If you are not familiar with ClearCase, it is recommended to consult with your ClearCase administrator while configuring the integration.

Set Up ClearCase Base Snapshot View Repository

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.

3. Select **ClearCase** from the drop-down menu and click **Set**.
4. Configure repository:
 - **Name** the repository. This is the name AnthillPro will use to identify this repository. This name does not correspond to the actual repository, and is simply an identifying label used by the AnthillPro system.
 - **Description**. Give an optional description.
 - **Command Path**. Provide the location of the ClearCase executable if it is not in the system path.
5. Click **Set**.
6. Select the **Security** tab and click the **Edit** button. Determine **permissions**, click **Save**, and then click **Done**.
7. Click **Done**.

ClearCase Base Snapshot View Source Configuration

Once the main ClearCase repository is identified (see Set Up ClearCase Base Snapshot View Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the ClearCase repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Configure, or Edit Source, link under the Source Config menu on the Workflow Main page.)

1. Once ClearCase is set up with AnthillPro (see Set Up ClearCase Base Snapshot View Repository), create a project.
2. Select the Mode and View Type once ClearCase has been selected as the repository type.
 - **Mode**. Choose **Base**.
 - **View Type**. Choose **Snapshot**.
 - Click **Select**.
3. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see Create a New Workflow before continuing. Configure the source on the workflow:
 - **Load Rules**. Give the Load Rules configured in the View Configuration Specifications to tell AnthillPro what projects, branches, and labels to use. AnthillPro requires the specs to be in the form `<vob-name><path>:<branch>:<label>`. Each Load Rule must be input on a separate line.

The Load Rules also support AnthillPro properties. E.g., `${property:vob.tag}:/:int_dc_10.02`.

If the View Load Rules are configured with the value `BaseModeTest:Anthill-Example::`, then they are input as `vob-name BaseModeTest` (following the form `<vob-name><path>:<branch>:<label>`) in the Load Rules field. Note that the Source Config deals with the `vob-name BaseModeTest` and that the project is located in the `Anthill-Example` directory under our VOB.

To build from the branch named `test`, the Load Rule would be `BaseModeTest:Anthill-Example:test:` (all builds performed on the test branch).

To perform a build on label `1.0.3` the Load Rule would be `BaseModeTest:Anthill-Example::1.0.3`.

To build from label 1.0.3 of the branch named test, the Load Rule would be `BaseMode-Test:Anthill-Example:test:1.0.3`.

Example Load Rule Configuration

For the load rules below, AnthillPro needs to know the path and optional branch and label the configspec is checking out in order to find the changes and then apply any needed labels. For example, the branch and label would come from lines starting with `element`, paths come from the lines starting with `load`, etc.

```
element* CHECKEDOUT

element/elementName/... ../elementNameApplication/LATEST
element/elementName/... ../iter2_elementNameApplication/LATEST -mkbranch elementNameApp
element/mmisPortal/... /main/LATEST -mkbranch iter2_portal
element* ../element_customization/LATEST
element* ../main/LATEST
load /elementNamePortal
load /elementNameCommon
load /elementNameFramework
```

For example, if you look at the first line in the load rules:

```
element/elementName/... ../elementNameApplication/LATEST
```

means that for path `/elementName` you need to get the code from the `elementNameApplication` branch (LATEST is the default label so you can ignore it when configuring AnthillPro). This translates in the following load rule (using the pattern `VOB:/path_in_vob:branch:label`):

```
elementName://elementNameApplication:
```

In the same manner, the second line translates into

```
elementName://iter2_elementNameApplication:
```

However, the `-mkbranch` in the line is ignored when configuring AnthillPro since `-mkbranch` says it's not about getting code but committing code back.

For the line `element* ../element_customization/LATEST` means that for every load rule we need to get the code from the `element_customization` branch. This requires one load rule be configured in AnthillPro per load rule in ClearCase. Thus, you would have something like:

```
elementName://element_customization:
```

```
elementNameCommon://element_customization:
```

```
elementNameFramework://element_customization:
```

Note that `element* ../main/LATEST` also needs to be addressed. In the example, you will need to get code from the `main` branch. This should look something like:

```
elementNamePortal://:
```

```
elementNameCommon://:
```

```
elementNameFramework://:
```

- **Create global labels.** Check the box to create global labels. You can also specify in which VOBs those labels will be created (see below). If you do not specify any VOBs (below), the Load Rules will be used to determine all the VOBs that are used, and the labels will be created in those VOBs.
- **Create labels in VOBs.** Please list the names of the VOBs (one per line) where the global labels should be created. If you are not creating global labels (option above is not selected) you can list all the VOBs used in your config spec or leave the field empty, and the Load Rules will be used to determine where the labels need to be created.
- **Host name.** Provides the value for the `-host` parameter to be used in the `mkview` command. Use this field only if the view and storage are in different locations.
- **Global storage path (gpath).** Provides the value for the `-gpath` parameter to be used in the `mkview` command. Use this field only if the view and storage are in different locations.
- **Host storage path (hpath).** Provides the value for the `-hpath` parameter to be used in the `mkview` command. Use this field only if the view and storage are in different locations.
- **View Strategy.** Select a view strategy:
 - *Create a new view every time we build.* If this option is chosen, AnthillPro will check for an existing snapshot view and remove it before creating a new view with every build. This option will ensure that the view always represents the state of the current build. Proceed to Item 4.
 - *Create a new view only if one does not exist already.* AnthillPro will check for an existing snapshot view. If one is found, a new view will not be created. If no existing view is found, AnthillPro will create a new one. This option allows you to use an existing view; however, that view may not represent the state of the latest build. Proceed to Item 5.
 - *The view already exists and will be used every time.* AnthillPro will not create a new view and use the existing view. This option will always use the view that previously exists. If no view exists, one must be created prior to using this option. Proceed to Item 6.

4. Create a new view every time we build View Strategy. Configure:

- **Working Directory Script.** Select the Working Directory Script from the drop-down menu. The working directory is the location where the agent is going to run its commands. For example, if the working directory is `C:\projects\Project_A\` every command AnthillPro issues will be from that specific directory.

If no Working Directory Script is configured in the job, the script selected here will be used. See Working Directory Scripts.

- **View Storage Directory.** Give the location of the view on the local machine. On Windows this will be a of the form `//host/share/view_storage`.
- **Project Name.** Give the name to be used in creating the view name.
- **Config Spec.** Provide the config spec to be used for the new view. Please replace any labels used in the config spec with a token.

For example: `/main/LATEST` should become `/main/${LATEST}` and `/main/test/1.0.3` should become `/main/test/${1.0.3}`.

Place a time token after any element rules that do not load from a particular label: `element * /main/LATEST` should become `element * /main/LATEST ${time.token}` and `element * /main/test/LATEST` should become `element * /main/test/LATEST ${time.token}`.

Do not assign the `element * /main/test/1.0.3` a time token.

- **Check New Configspec.** Select here if you want to compare the newly set config spec against the user specified config spec. This option allows you to determine success based on the new config spec set on the view rather than the outcome of the command.
- **Label Script.** Give the label to check out source from (if any).
- **Use Tags.** Select here if you want your view to be globally created. Having this option checked can cause problems with creating and dropping snapshots if you wish to have the same snapshot on multiple agents.
- Proceed to Item 7.

5. **Create a new view only if one does not exist already** View Strategy. Configure:

- **Working Directory Script.** Select the Working Directory Script from the drop-down menu. The working directory is the location where the agent is going to run its commands. For example, if the working directory is `C:\projects\Project_A\` every command AnthillPro issues will be from that specific directory.

If no Working Directory Script is configured in the job, the script selected here will be used. See Working Directory Scripts.

- **View Storage Directory.** Give the location of the view on the local machine. On Windows this will be a UNC path of the form `//host/share/view_storage`.
- **Project Name.** Give the name to be used in creating the view name.
- **Config Spec.** Provide the config spec to be used for the new view. Please replace any labels used in the config spec with a token.

For example: `/main/LATEST` should become `/main/${LATEST}` and `/main/test/1.0.3` should become `/main/test/${1.0.3}`.

Place a time token after any element rules that do not load from a particular label: `element * /main/LATEST` should become `element * /main/LATEST ${time.token}` and `element * /main/test/LATEST` should become `element * /main/test/LATEST ${time.token}`.

Do not assign the `element * /main/test/1.0.3` a time token.

- **Check New Configspec.** Select here if you want to compare the newly set config spec against the user specified config spec. This option allows you to determine if success was based on the new config spec set on the view rather than the outcome of the command.
- **Label Script.** Give the label to check out source from (if any).
- **Use Tags.** Select here if you want your view to be globally created. Having this option checked can cause problems with creating and dropping snapshots if you wish to have the same snapshot on multiple agents.
- **Delete view-private files during cleanup.** Select here if you would like AnthillPro to detect and delete any view-private files in the local view. This is highly recommended unless your build script handles cleanup of the view. Only the paths specified in the Load Rules above will be checked for view-private files.
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: **/ -
- for any number of any characters except \ and /: * -
- for any single character / including \ and / Changelog file paths containing drive letter or any prefixes: ? -

Make sure to include the * at the beginning of the path.

Example exclude paths: -**/abc/**/* Except for: +**/abc/*.html. This will exclude everything under the abc directory except for the HTML files in it.

- Proceed to Item 7.

6. The view already exists and will be used every time View Strategy. Configure:

- **View Storage Directory.** Give the location of the view on the local machine. On Windows this will be a UNC path of the form //host/share/view_storage.
- **Delete view-private files during cleanup.** Select here if you would like AnthillPro to detect and delete any view-private files in the local view. This is highly recommended unless your build script handles cleanup of the view. Only the paths specified in the Load Rules above will be checked for view-private files.
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: **/ -
- for any number of any characters except \ and /: * -
- for any single character / including \ and / Changelog file paths containing drive letter or any prefixes: ? -

Make sure to include the * at the beginning of the path.

Example exclude paths: -**/abc/**/* Except for: +**/abc/*.html. This will exclude everything under the abc directory except for the HTML files in it.

- Proceed to Item 7.

7. Click Save.

ClearCase UCM Dynamic View

The first step in using a ClearCase UCM Dynamic View repository with AnthillPro is to configure it by following the Repositories link on the System page. The configuration will allow basic information regarding ClearCase to be reused by several project workflows. Once configured, the repository will be listed on the Repositories main page.

Once the main repository has been identified, your projects can then use the ClearCase repository. During project creation, associate the workflow with a specific ClearCase repository and determine the mode, view, and specify the Load Rules. See ClearCase UCM Dynamic View Source Configuration.

See also ClearQuest.

ClearCase UCM Dynamic View Prerequisites

- ClearCase must already be installed. See ClearCase documentation [<http://www-01.ibm.com/support/docview.wss?rs=984&uid=swg21239261>].
- You must have administrative permissions. See Manage Security.
- The cleartool executable and the Load Rules must be available to complete the configuration.
- If you are not familiar with ClearCase, it is recommended to consult with your ClearCase administrator while configuring the integration.

Set Up ClearCase UCM Dynamic View Repository

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **ClearCase** from the drop-down menu and click **Set**.
4. Configure:
 - **Name** the repository. This is the name AnthillPro will use to identify this repository. This name does not correspond to the actual repository, and is simply an identifying label used by the AnthillPro system.
 - **Description**. Give an optional description.
 - **Command Path**. Provide the location of the ClearCase executable if it is not in the system path.
5. Click **Save**.
6. If not setting a **Repository Trigger** or **Security**, click **Set** then **Done** to complete. Otherwise proceed to **item 7** to set a trigger or **item 9** to set security.
7. Select the **Trigger** tab. To either **deactivate** or **delete** the repo trigger, click the appropriate button. If the repo trigger is deactivated/deleted, you will need to configure a workflow trigger for every project.
8. If not setting **Security**, click **Activate** then **Done** to complete. Otherwise click **Activate** and proceed to **item 9**.
9. Select the **Security** tab and click the **Edit** button. Check the appropriate boxes to determine user-role access (See Manage Security), and click **Save**.
- 10 Click **Done**.

ClearCase UCM Dynamic View Source Configuration

Once the main ClearCase repository is identified (see Set Up ClearCase UCM Dynamic View Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the ClearCase repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Configure, or Edit Source, link under the Source Config menu on the Workflow Main page.)

1. Once ClearCase is set up with AnthillPro (see Set Up ClearCase UCM Dynamic View Repository), create a

project.

2. Select the Mode and View Type once ClearCase has been selected as the repository type.

- **Mode.** Choose **UCM**.
- **View Type.** Choose **Use Dynamic**.
- Click **Select**.

3. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see [Create a New Workflow](#) before continuing. Configure the source on the workflow:

- **View Location.** Give the location of the view on the local machine (AnthillPro Agent). For Windows this will be a UNC path of the form `//host/share/view_storage`.
- **View Name.** If the rebase step is used, you must give the view name here. Otherwise, leave this field blank.
- **Stream Name.** Give the ClearCase stream name that is used by this view.

Check the box if AnthillPro is to build the child stream. If the field is inactive, you cannot build child streams.

- **Project VOB Name.** Give the project VOB name.
- **Paths.** Give the paths to tell AnthillPro what projects, branches, and labels to use. A view in ClearCase can contain code from any number of VOBs. By giving the paths (each on its own line) you are telling AnthillPro which directories in which VOBs you want AnthillPro to use when checking for changes and when creating a baseline. AnthillPro requires the specs to be in the form `vob/path/to/files`. For example, to perform a build on label 1.0.3, the Load Rule would be `VOB/BaseModeTest/Anthill-Example/1.0.3`.

To build from label 1.0.3 of the branch named test, the Load Rule would be `VOB/BaseMode-Test/Anthill-Example/test/1.0.3`.

- **Create global labels.** Check the box to create global labels. You can also specify in which VOBs those labels will be created (see below). If you do not specify any VOBs (below), the Load Rules will be used to determine all the VOBs that are used and the labels will be created in those VOBs.
- **Create labels in VOBs.** Please list the names of the VOBs (one per line) where the global labels should be created. If you are not creating global labels (option above is not selected) you can list all the VOBs used in your config spec or leave the field empty and the Load Rules from above will be used to determine where the labels need to be created in order to complete the labeling process successfully.
- **Delete view-private files during cleanup.** Select here if you would like AnthillPro to detect and delete any view-private files in the local view. This is highly recommended unless your build script handles cleanup of the view. Only the paths specified in the Load Rules will be checked for view-private files.
- **Lock Objects.** Please list the names of the objects (one per line) that you would like locked/unlocked during the build process. Use the following format: `type:Name@VOB` (or refer to the lock/unlock ClearCase command reference for more details). Be advised that if you leave this field empty, there is nothing to prevent an update of the dynamic view while a build is in progress.
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: **/ -
- for any number of any characters except \ and /: * -
- for any single character / including \ and / Changelog file paths containing drive letter or any prefixes: ? -

Make sure to include the * at the beginning of the path.

Example exclude paths: -**/abc/**/* Except for: +**/abc/*.html. This will exclude everything under the abc directory except for the HTML files in it.

4. Click **Save**.

Using Triggers with ClearCase UCM Dynamic View

ClearCase UCM repository triggers allow you to create a single hook in the repository that is capable of triggering every workflow using this repository. For deliver events in your ClearCase repository, use Perl or a utility like wget or curl to generate an HTTP post to the URL specified in the trigger. The examples below are for shell scripts. The 'xxxx' values for pvob-name and stream-name should be replaced with the project VOB name and stream name that was delivered to.

By default, AnthillPro activates the repository trigger when the repository is configured. When configuring an AnthillPro project, the repo trigger is automatically generated for every workflow.

Using wget in a Unix/Linux shell script:

```
#!/bin/bash

TRIGGER_URL="http://localhost:8080/trigger"
CODE=f3b5110446797dbce6579d01fe55f5854ee370a2

wget --tries 1 --no-check-certificate "$TRIGGER_URL" \
  --post-data="code=$CODE&pvob-name=xxxx&stream-name=xxxx" -O /dev/null
```

Using wget in a Windows shell script:

```
@echo off
setlocal

set TRIGGER_URL=http://localhost:8080/trigger
set CODE=f3b5110446797dbce6579d01fe55f5854ee370a2

wget --tries 1 --no-check-certificate "http://localhost:8080/trigger"
--post-data="code=%CODE%&pvob-name=xxxx&stream-name=xxxx"
```

Using curl in a Unix/Linux shell script:

```
#!/bin/bash

TRIGGER_URL="http://localhost:8080/trigger"
CODE=f3b5110446797dbce6579d01fe55f5854ee370a2

curl --retry 1 -k -d "code=$CODE&pvob-name=xxxx&stream-name=xxxx" -o /dev/null "$TRIGGER_U
```

Using curl in a Windows shell script:

```
@echo off
setlocal

set TRIGGER_URL=http://localhost:8080/trigger
set CODE=f3b5110446797dbce6579d01fe55f5854ee370a2

curl --retry 1 -k -d "code=%CODE%&pvob-name=xxxx&stream-name=xxxx" -o /dev/null "%TRIGGER_
```

ClearCase UCM Snapshot View

The first step in using a ClearCase UCM Snapshot View repository with AnthillPro is to configure it by following the Repositories link on the System page. The configuration will allow basic information regarding ClearCase to be reused by several projects. Once configured, the repository will be listed on the Repositories main page.

Once the main repository has been identified, your workflows can then use the ClearCase repository. During project creation, associate the project workflow with a specific ClearCase repository and determine the mode, view, and specify the Load Rules. See ClearCase UCM Snapshot View Source Configuration.

See also ClearQuest.

ClearCase UCM Snapshot View Prerequisites

- ClearCase must already be installed. See ClearCase documentation [<http://www-01.ibm.com/support/docview.wss?rs=984&uid=swg21239261>].
- You must have administrative permissions. See Manage Security.
- The cleartool executable and the Load Rules must be available to complete the configuration.
- If you are not familiar with ClearCase, it is recommended to consult with your ClearCase administrator while configuring the integration.

Set Up ClearCase UCM Snapshot View Repository

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **ClearCase** from the drop-down menu and click **Set**.
4. Configure:
 - **Name** the repository. This is the name AnthillPro will use to identify this repository. This name does not correspond to the actual repository, and is simply an identifying label used by the AnthillPro system.
 - **Description**. Give an optional description.
 - **Command Path**. Provide the location of the ClearCase executable if it is not in the system path.
5. Click **Save**.
6. If not setting a **Repository Trigger** or **Security**, click **Set** then **Done** to complete. Otherwise proceed to **item 7** to set a trigger or **item 9** to set security.
7. Select the **Trigger** tab. To either **deactivate** or **delete** the repo trigger, click the appropriate button. If the repo

trigger is deactivated/deleted, you will need to configure a workflow trigger for every project.

8. If not setting **Security**, click **Activate** then **Done** to complete. Otherwise click **Activate** and proceed to **item 9**.
9. Select the **Security** tab and click the **Edit** button. Check the appropriate boxes to determine user-role access (See Manage Security), and click **Save**.
- 10 Click **Done**.

ClearCase UCM Snapshot View Source Configuration

Once the main ClearCase repository is identified (see Set Up ClearCase UCM Snapshot View Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the ClearCase repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Configure, or Edit Source, link under the Source Config menu on the Workflow Main page.)

During source configuration, you must select a View Strategy. The strategy will determine how AnthillPro interacts with existing views and determine when AnthillPro should create a new view. Select a strategy that either (*a.*) creates a new view with every build; (*b.*) create a new view if and only if a view does not exist; or (*c.*) does not create a new view and uses an existing view. Only one view may be configured on an individual workflow; however, if a project has multiple workflows, a separate View Strategy may be used for each workflow. See View Strategy.

1. Once ClearCase is set up with AnthillPro (see Set Up ClearCase UCM Snapshot View Repository), create a new project.
2. Select the Mode and View Type once ClearCase has been selected as the repository type.
 - **Mode.** Choose **UCM**.
 - **View Type.** Choose **Snapshot**.
 - Click **Select**.
3. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see Create a New Workflow before continuing. Configure the source on the workflow:
 - **View Location.** Give the location of the view on the local (Agent) machine. On Windows this will be a UNC path of the form //host/share/view_storage.
 - **View Name.** If the rebase step is used, you must give the view name here. Otherwise, leave this field blank.
 - **Stream Name.** Give the ClearCase stream name that is used by this view.
 - **Project VOB Name.** Give the project VOB name.
 - **Paths.** Give the paths to tell AnthillPro what projects, branches, and labels to use. A view in ClearCase can contain code from any number of VOBs. By giving the paths (each on its own line) you are telling AnthillPro which directories in which VOBs you want AnthillPro to use when checking for changes and when creating a baseline. AnthillPro requires the specs to be in the form vob/path/to/files. For example, to perform a build on label 1.0.3, the Load Rule would be VOB/BaseModeTest/Anthill-Example/1.0.3.

To build from label 1.0.3 of the branch named test, the Load Rule would be VOB/BaseMode-Test/Anthill-Example/test/1.0.3.
 - **Is building in child stream.** Check the box if AnthillPro is to build the child stream. If the field is inactive, you cannot build child streams. Only select this option if you are using a child stream off of the main develop-

ment stream and nothing is delivered to this stream. This option tells AnthillPro to look for changes using the baseline activity set rather than looking for changes on the stream itself.

- **View Strategy.** Select a view strategy:
 - *Create a new view every time we build.* If this option is chosen, AnthillPro will check for an existing snapshot view and remove it before creating a new view with every build. This option will ensure that the view always represents the state of the current build. Proceed to Item 4.
 - *Create a new view only if one does not exist already.* AnthillPro will check for an existing snapshot view. If one is found, a new view will not be created. If no existing view is found, AnthillPro will create a new one. This option allows you to use an existing view; however, that view may not represent the state of the latest build. Proceed to Item 5.
 - *The view already exists and will be used every time.* AnthillPro will not create a new view and use the existing view. This option will always use the view that previously exists. If no view exists, one must be created prior to using this option. Proceed to Item 6.

4. **Create a new view every time we build** View Strategy. Configure:

- **View Storage Directory.** Give the location of the view on the local machine. On Windows this will be a of the form `//host/share/view_storage`.
- **Host name.** Provides the value for the `-host` parameter to be used in the `mkview` command. Use this filed only if the view and storage are in different locations.
- **Global storage path (gpath).** Provides the value for the `-gpath` parameter to be used in the `mkview` command. Use this filed only if the view and storage are in different locations.
- **Host storage path (hpath).** Provides the value for the `-hpath` parameter to be used in the `mkview` command. Use this filed only if the view and storage are in different locations.
- **Use Tags.** Select here if you want your view to be globally created. Having this option checked can cause problems with creating and dropping snapshots if you wish to have the same snapshot on multiple agents.
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be proceeded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/` -
- for any number of any characters except `\` and `/`: `*` -
- for any single character / including `\` and / Changelog file paths containing drive letter or any prefixes: `?` -

Make sure to include the `*` at the beginning of the path.

Example exclude paths: `-**/abc/**/*` Except for: `+**/abc/*.html`. This will exclude everything under the abc directory except for the HTML files in it.

- Proceed to Item 7.

5. **Create a new view only if one does not exist already** View Strategy. Configure:

- **View Storage Directory.** Give the location of the view on the local machine. On Windows this will be a UNC path of the form `//host/share/view_storage`.
- **Host name.** Provides the value for the `-host` parameter to be used in the `mkview` command. Use this field only if the view and storage are in different locations.
- **Global storage path (gpath).** Provides the value for the `-gpath` parameter to be used in the `mkview` command. Use this field only if the view and storage are in different locations.
- **Host storage path (hpath).** Provides the value for the `-hpath` parameter to be used in the `mkview` command. Use this field only if the view and storage are in different locations.
- **Use Tags.** Select here if you want your view to be globally created. Having this option checked can cause problems with creating and dropping snapshots if you wish to have the same snapshot on multiple agents.
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by `+` or `-` to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/` -
- for any number of any characters except `\` and `/`: `*` -
- for any single character / including `\` and / Changelog file paths containing drive letter or any prefixes: `?` -

Make sure to include the `*` at the beginning of the path.

Example exclude paths: `-**/abc/**/*` Except for: `+**/abc/*.html`. This will exclude everything under the `abc` directory except for the HTML files in it.

- Proceed to Item 7.

6. The view already exists and will be used every time View Strategy. Configure:

- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by `+` or `-` to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/` -
- for any number of any characters except `\` and `/`: `*` -
- for any single character / including `\` and / Changelog file paths containing drive letter or any prefixes: `?` -

Make sure to include the `*` at the beginning of the path.

Example exclude paths: `-**/abc/**/*` Except for: `+**/abc/*.html`. This will exclude everything under the `abc` directory except for the HTML files in it.

- Proceed to Item 7.

7. Click **Save**.

Using Triggers with ClearCase UCM Snapshot View

ClearCase UCM repository triggers allow you to create a single hook in the repository that is capable of triggering every workflow using this repository. For deliver events in your ClearCase repository, use Perl or a utility like wget or curl to generate an HTTP post to the URL specified in the trigger. The examples below are for shell scripts. The 'xxxx' values for pvob-name and stream-name should be replaced with the project VOB name and stream name that was delivered to.

By default, AnthillPro activates the repository trigger when the repository is configured. When configuring an Ant-hillPro project, the repo trigger is automatically generated for every workflow.

Using wget in a Unix/Linux shell script:

```
#!/bin/bash

TRIGGER_URL="http://localhost:8080/trigger"
CODE=f3b5110446797dbce6579d01fe55f5854ee370a2

wget --tries 1 --no-check-certificate "$TRIGGER_URL" \
  --post-data="code=$CODE&pvob-name=xxxx&stream-name=xxxx" -O /dev/null
```

Using wget in a Windows shell script:

```
@echo off
setlocal

set TRIGGER_URL=http://localhost:8080/trigger
set CODE=f3b5110446797dbce6579d01fe55f5854ee370a2

wget --tries 1 --no-check-certificate "http://localhost:8080/trigger"
--post-data="code=%CODE%&pvob-name=xxxx&stream-name=xxxx"
```

Using curl in a Unix/Linux shell script:

```
#!/bin/bash

TRIGGER_URL="http://localhost:8080/trigger"
CODE=f3b5110446797dbce6579d01fe55f5854ee370a2

curl --retry 1 -k -d "code=$CODE&pvob-name=xxxx&stream-name=xxxx" -o /dev/null "$TRIGGER_U
```

Using curl in a Windows shell script:

```
@echo off
setlocal

set TRIGGER_URL=http://localhost:8080/trigger
set CODE=f3b5110446797dbce6579d01fe55f5854ee370a2

curl --retry 1 -k -d "code=%CODE%&pvob-name=xxxx&stream-name=xxxx" -o /dev/null "%TRIGGER_U
```

ClearCase Builds

If you use the Job Wizard to create a build job, only the generic job steps are used -- which should suffice for most

builds. However, the integration also includes ClearCase-specific steps (commands, etc.) that you can add to your build job. While these steps are not required for a basic build, they may be helpful as you design more complicated build jobs.

When setting up a build and not using the Wizard, most jobs will contain the following steps:

1. **Cleanup Step.** Cleans up the workspace prior to populating it. This ensures that any files you might pick up from the build actually belong to the build and not a previous build.
2. **Populate Workspace.** Places the checked-out code (defined in the workflow's source configuration) in the workspace.
3. **Get Changelog.** The retrieved changelog is usually based on the changes made since the previous build. This step enables AnthillPro to extract data from the SCM and then store it in the AnthillPro data warehouse. Since AnthillPro stores the changelogs, it can parse the data, allowing you to override the default behavior. For example, you can select a starting point for the changelog based on criteria such as the latest production build.
4. **Stamp.** Applies a stamp to the build, which is used to identify the build within AnthillPro.
5. **Get Dependency Artifacts.** If your project depends on any other project, you need to specify it here. Otherwise, this step will just be skipped if no dependencies are defined.
6. **Builder.** Usually points to the location of tool or script used for the build.
7. **Label Source.** AnthillPro can also apply a label to the source code used in the build (e.g., snapshot, baseline). This unique identifier for a build can be used to recreate a build if necessary.
8. **Assign Status - Success.** Applies the status when the build completes successfully.
9. **Assign Status - Failure.** Applies the status when the build fails.

In addition, the ClearCase integration includes tool-specific job steps that may be added to the generic build job or used to replace one of the generic steps where appropriate. When configuring, the steps are available under the SCM folder:

- **ClearCase Changelog.** Performs a ClearCase changelog.
- **ClearCase Cleanup.** Perform a cleanup of the current ClearCase working directory.
- **ClearCase Label.** ClearCase-specific label step.
- **Lock (ClearCase Base Dynamic Plugin only).** Locks ClearCase resources.
- **Unlock (ClearCase Base Dynamic Plugin only).** Unlocks ClearCase resources.
- **ClearCase Populate Workspace (Legacy integration only).** Populates the workspace With source from ClearCase.

CVS

The first step in using a CVS repository with AnthillPro is to configure it by following the Repositories link on the System page. The configuration will allow basic information regarding CVS to be reused by several project workflows. Once configured, the repository will be listed on the Repositories main page.

Once the main repository has been identified, your workflows can then use the CVS repository. During project cre-

ation, associate the originating workflow with a specific CVS repository. See CVS Source Configuration.

CVS Prerequisites

- You must have read and write permissions to the System page. See Manage Security.
- A CVS repository must already be created. See CVS documentation [<http://ximbiot.com/cvs/manual/>].
- In order to use ViewVC integration, the Source Configuration **Repository Name** field must be the actual name. It is not possible to use ViewVC if properties are set for the **Repository Name** field. See ViewVC.

Set Up CVS Repository

Configure AnthillPro with an existing CVS repository. This example covers the configuration of a CVS repository for the Anthill Example project, as well as covering some of the variations needed to complete a configuration.

1. Go to **System > Repositories** from the **Project Support** menu.
2. Click the **Create New** button on the **Repositories** page.
3. Select **CVS** from the drop-down menu and click **Set**.
4. Configure the repository.
 - **Name.** This is the name AnthillPro will use to identify this repository. This name does not correspond to the actual repository, and is simply an identifying label used by the AnthillPro system.
 - **Description.** Provide description (optional).
 - **CVSROOT.** The CVSROOT tells AnthillPro the authentication method, the user name, the domain name, and the repository location. An example CVSROOT would be:
`:pserver:anthill-example@cvs2.urbancode.com:/usr/local/anthill-test:`
 - `:pserver` is the authentication method you are using to connect to the repository, and specifies that you are using a password server. If you omit this from the CVSROOT, CVS will use the RSH default.
 - `:anthill-example` is the user name we want connected.
 - `@cvs2.urbancode.com` is the machine host (domain) name.
 - `:/usr/local/anthill-test` is the CVS repository location on the server.

If using an external protocol for authentication, specify `:ext`. By specifying `:ext`, you can login with any authentication protocol for which a client can respond to the same command options/api as RSH.

If you are using a fork or local server, you need to specify `:local` or `:fork` in the CVSROOT.

- **CVS_RSH.** Specify the remote shell used to connect to the repository. Enter a value in this field to set the CVS_RSH environment variable. Use this field to also tell the CVS client which protocol to use when logging into the repository.
- **CVS Password.** Provide the password corresponding to the account specified in the CVSROOT. (You will need to specify a password if you are using Kerberos or `:pserver` for the Anthill-Example project. You can use a `pub_key/priv_key` to handle authentication with RSH and SS. This means that you do not need to enter a password when authenticating in this fashion. It is also possible to perform a login with `:sspi`, Windows'

single sign-on, which does not require a password [because authentication is handled with Active Directory when logging into Windows]).

Select either: No Password, Use Password, or Use Script.

- **No Password.** If no password is associated with AnthillPro's CVS account, use this option.
- **Use Password.** If a password is associate with AnthillPro's CVS account, select this potion. Give the password in the Password field and confirm the password.

Check the **Use in CVSROOT** box to add the password in the CVSROOT. The password will be hidden in all logs files. This is a workaround if CVS is unable to read the .cvspass file in certain scenarios.

- **Use Script (advanced).** Give a parameterizable value that will resolve to the password. Use this instead of the Password field if the password will be stored in a secure property resolved at runtime. See Scripting.

Check the **Use in CVSROOT** box to add the password in the CVSROOT. The password will be hidden in all logs files. This is a workaround if CVS is unable to read the .cvspass file in certain scenarios.

- **Command Path.** The location of the CVS executable, if not in the system path. If the CVS executable was not on the command path (as it was in the example), you would type it in the Command Path text field. If you do not want to ensure the executable is on the path of every agent, then you must set an agent property for every agent (in AnthillPro) with the path to the executable, and then reference that property in the Command Path field.
 - In a distributed environment, it becomes very difficult to put the executable on every agent path.
- **Repository Name.** Give the name of the repository.

If using the ViewVC integration, the Repository Name field must be the ViewVC repository name. It is not possible to use ViewVC if properties are set for the Repository Name field. See ViewVC.

- **Source Viewer.** Select the Source Viewer Type from the drop-down menu (if using the ViewVC integration). Otherwise select **None**. See ViewVC.
- **Is CvsNT.** Select here if you are using a CvsNT repository. CvsNT cannot authenticate using AnthillPro's normal means of password specification (writing the password and encrypting it to a .cvspass file in the user home directory). For example, on Windows, using CvsNT with :pserver will not work. By checking 'Is CvsNT', AnthillPro will run a CVS login command during every step. See CvsNT documentation [<http://www.cvsnt.org/manual/html/>].
- **Place Password in CVSROOT.** By selecting this option, you require the password to be added into the CVSROOT and to be hidden in the logs files. This is a workaround because CVS is unable to read the .cvspass file in certain scenarios. When printed out, the ENV_VARIABLES hide the password.
- **Date Format.** Give the date format to use when parsing change logs. The default is yyyy/MM/dd HH:mm:ss, which should work with older versions of CVS. If using a newer versions of CVS use: yyyy-MM-dd HH:mm:ss. See CVS documentation [<http://ximbiot.com/cvs/manual/>].
- Click **Set**.

5. Select the **Security** tab and click the **Edit** button. Determine **permissions** and click **Save**. Click **Done**. See Manage Security.
6. Click **Done**.

CVS Repository Trigger

CVS configuration is contained in a special module called CVSROOT. You must first configure a CVS repository and then checkout this module before you can continue.

The file you are interested in is called *loginfo*. It has a special format that must be followed. Each line contains a regular expression that matches the module repository path, a space, and a shell command. The default file CVS provides contains comments that explains this in more detail.

In the command, you use a utility like Wget to generate an HTTP request to the URL specified in the trigger:

```
^MyProject (wget -t 0 https://server/trigger
  --post-data='triggerId=31&code=95cde532437151c551cf062bf93c0d12de9209c7'
  -O - &>/dev/null)
```

A nested module would be handled like this:

```
^MyProject/SubProject (wget -t 0 https://server/trigger
  --post-data='triggerId=31&code=95cde532437151c551cf062bf93c0d12de9209c7'
  -O - &>/dev/null)
```

CVS Source Configuration

Once the main CVS repository is identified (see Set Up CVS Repository), configure the specific repository a workflow uses. During project creation the originating workflow is first associated with the CVS repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Configure, or Edit Source, link under the Source Config menu on the Workflow Main page.)

1. Once CVS is set up with AnthillPro (see Set Up CVS Repository), create a new project.
2. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see Create a New Workflow before continuing. Configure the source on the workflow:

- **Working Directory Script.** Select the Working Directory Script from the drop-down menu. The working directory is the location where the agent is going to run its commands. For example, if the working directory is `C:\projects\Project_A\` every command AnthillPro issues will be from that specific directory.

If no Working Directory Script is configured in the job, the script selected here will be used. See Working Directory Scripts.

- **Module Name.** Give the CVS module name to be retrieved by this workflow. See CVS documentation [<http://ximbiot.com/cvs/manual/>].
- **Branch.** Give the CVS branch name to be retrieved by this workflow. See CVS documentation [<http://ximbiot.com/cvs/manual/>].
- **Tag.** If using tags, give the tag to use for checkout. This may include variables passed to the workflow via `${property:Name}` format. See Scripting.
- **Directory Offset.** Give the sub-directory where this module is to be placed within the working directory. If the root of the working directory is to be used, leave this item blank.
- **Local Folder Only.** Check the box to only retrieve files from the specified folder (for non-recursive checkout).
- **Prune Empty Directories.** Check the box to clean up empty directories. If selected, the Prune Empty Directories option will delete any empty directories during checkout.

- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: ** / -
- for any number of any characters except \ and /: * -
- for any single character / including \ and / Changelog file paths containing drive letter or any prefixes: ? -

Make sure to include the * at the beginning of the path.

Example exclude paths: -**/abc/**/* Except for: +**/abc/*.html. This will exclude everything under the abc directory except for the HTML files in it.

3. Click **Save**.

Dimensions

The first step in using a Serena Dimensions repository with AnthillPro is to configure it from the Repositories item on the Administration page (see Set Up Dimensions Repository). This configuration will allow basic information about where the Dimensions repository is, and how to connect to it to be reused by several project workflows. Once the main repository has been identified, your projects can then use the Dimensions repository during workflow source configuration (see Dimensions Source Configuration). During project creation, the workflow is associated with a specific Dimensions repository.

Dimensions Prerequisites

- Dimensions must already be installed.
- You must have administrative permissions. See Manage Security.

Set Up Dimensions Repository

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **Dimensions** from the drop-down menu and click **Set**.
4. Configure repository:
 - **Name** the repository. This name is used throughout AnthillPro. For example, the name given here will be used during Source Configuration.
 - **Description.** Give an optional description.
 - **DMCLI Command Path.** Give the usual command path to the dimensions command line utility. This field is

only necessary if the location of the `dmcli` executable, if not in system path.

- **Download Command Path.** Give the location of the `download` executable, if not in system path. In order to make the `populate` command sane, you can use another Dimensions package, -- the Dimensions Make Package -- which provides the client with subsequent commands for download and upload. In order to use the Dimensions driver, have the Dimensions Make Package installed and on the path, or provide the path to it here.
- **User Name.** Provide a Dimensions user name for accessing Dimensions database. We recommend creating a user exclusively for AnthillPro. This allows for clear traceability, security management, and easier transition as the team changes.
- **Password.** Give the user's password. Use either Set Password, or Use Script:
 - **Set Password.** Select this option to use the password associated with the AnthillPro user account. Input the password and confirm.
 - **Use Script (advanced).** Give a parameterizable value that will resolve to the password. Use this instead of the Password field if the password will be stored in a secure property (which is resolved at runtime). See Scripting.
- **Host Address.** Give the location of the Dimensions host.
- **Database Name.** Provide the name of the dimensions database.
- **Database Schema Name.** Give the database schema to connect to for the given database configured above.

5. Click **Set**.

6. Select the **Security** tab and click the **Edit** button. Determine **permissions**, click **Save**, and then click **Done**.

7. Click **Done**.

8. See Dimensions Source Configuration.

Dimensions Source Configuration

Once the main Dimensions repository is identified (see Set Up Dimensions Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the Dimensions repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Edit Source link on the Workflow Main page.)

1. Once Dimensions is set up with AnthillPro (see Set Up Dimensions Repository), create a project.
2. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see Create a New Workflow before continuing. Configure the source on the workflow:
 - **Workset Name.** Give the name of the workset to use. You may need to create a workset that frames the proper items to be built.

Note that if you do not provide a Workset, you must provide a Baseline. However, you cannot provide both.

- **Baseline Name.** Give the name of the baseline to build from. This could be a dynamic label like `#{property:baseline}` passed in at build time.

Note that if you do not provide a Baseline, you must provide a Workset. However, you cannot provide both.

- **Part Name.** Provide the name of the Part to use when labeling. This is the part name created by Dimensions.
- **Template Id.** Optional setting to use a Dimensions base-lining template when applying a baseline. If left empty, the `ALL_ITEMS_LATEST` is used which will apply the baseline to the most recent version of every item in the baseline.
- **Product Id.** Give the Dimensions project id (project name) for the project to be built.
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/` -
- for any number of any characters except `\` and `/`: `*` -
- for any single character / including `\` and `/` Changelog file paths containing drive letter or any prefixes: `?` -

Make sure to include the `*` at the beginning of the path.

Example exclude paths: `-**/abc/**/*` Except for: `+**/abc/*.html`. This will exclude everything under the abc directory except for the HTML files in it.

3. Click **Save**.

File System

The first step in using a File System repository with AnthillPro is to configure it by following the Repositories link on the System page. The configuration will allow basic information regarding the File System to be reused by several project workflows. Once configured, the repository will be listed on the Repositories main page.

Once the main repository has been identified, your workflows can then use the File System repository. During project creation, associate the workflow with a specific File System repository. See File System Source Configuration.

File System Prerequisites

- You must have administrative permissions. See Manage Security.
- The directory (file system) location must be available.

Configure File System Repository

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **File System** from the drop-down menu and click **Set**.

4. Configure repository:
 - **Name** the file system repository.
 - **Description.** Give an optional description.
5. Click **Set**.
6. Select the **Security** tab. See Manage Security.
7. Click **Edit**, determine **permissions**, click **Save**, and then click **Done**.
8. Click **Done**.

File System Source Configuration

Once the main File System repository is identified (see Configure File System Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the File System repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Configure, or Edit Source, link under the Source Config menu on the Workflow Main page.)

1. Once the File System is set up with AnthillPro (see Configure File System Repository), create a project.
2. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see Create a New Workflow before continuing. Configure the source on the workflow:
 - **Working Directory Script.** Select the Working Directory Script from the drop-down menu. The working directory is the location where the agent is going to run its commands. For example, if the working directory is `C:\projects\Project_A\` every command AnthillPro issues will be from that specific directory.

If no Working Directory Script is configured in the job, the script selected here will be used. See Working Directory Scripts.

- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/` -
- for any number of any characters except \ and /: `*` -
- for any single character / including \ and / Changelog file paths containing drive letter or any prefixes: `?` -

Make sure to include the `*` at the beginning of the path.

Example exclude paths: `-**/abc/**/*` Except for: `+/**/abc/*.html`. This will exclude everything under the abc directory except for the HTML files in it.

3. Click **Save**.

Use File System Repository

Because it is simple to set up, the File System Repository is typically used to perform builds on a local machine. Once configured (see Configure File System Repository), the File System is usually identified in the **Set Working Directory** step of the build job.

This section only covers the most common steps necessary to use a File System Repository.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. **Set Working Directory**. Click **Create Step**. Expand the **Miscellaneous** folder, select the **Set Working Directory** step, and click **Select**.
5. Select **New Script**.
 - **Name** the script.
 - **Description**. Provide a description.
 - **Script**. Give the working directory path. This can be hard coded (e.g., C:\directoryname) or include variables and scriptlets. See Scripting.

If a Set Working Directory job step is used, it will always override the Working Directory Script selected during Source Configuration (see File System Source Configuration). For example, adding a job step that sets the Working Directory to C:\Project_A\Subproject_01 will override the Working Directory (of C:\Project_A\) selected during Source Configuration. The job will always be run in the C:\Project_A\Subproject_01 directory. Before changing the Working Directory, see Working Directory Scripts.

- Click **Save**.
6. Select the newly created script from the drop-down menu.
 7. **Clean Working Directory**. Check the box to erases all files and subdirectories in the Working Directory when this step runs.

If selecting this option, ensure that the Working Directory is not set to something like C: or C:\ because **the entire contents of the C drive will be permanently removed**.

Also note that if the Working Directory of C:\Project_A\ is set to clean up the workspace, the contents of C:\Project_A\Subproject_01 will be removed as well. This may have an adverse effect on jobs that use the C:\Project_A\Subproject_01 directory.

8. **Show Additional Options (advanced)**. Select the Show Additional Options link to configure more options.
 - **Is Active**. Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Continuation Condition**. Select the condition which must be met for the process to continue (all steps pass; previous step passed; any step failed; always; or never).
 - **Ignore Failures**. Select Yes if this step should not affect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript**. Select a script for determining when commands should count as fail or succeed.

- **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

9. Click **Save**.

Git

The first step in using a Git repository with AnthillPro is to configure it by following the Repositories link on the System page. The configuration will allow basic information regarding Git to be reused by several project workflows. Once configured, the repository will be listed on the Repositories main page.

After the main repository has been identified, your workflows can then use the Git repository. During project creation, associate the project workflow with a specific Git repository and create a build job that includes the Git job steps. AnthillPro has four Git-specific steps that are used during job creation: **cleanup**, **populate workspace**, **get changelog**, and **label**. If you use the Job Wizard, AnthillPro will automatically add these steps during job configuration; however, if you create your own job, you will need to add these steps yourself (usually in the order given above).

AnthillPro and Cloned Git Repositories. When the configured workflow is run, AnthillPro will create a clone of the repository on the agent that is going to perform the build and check out a particular revision and then perform a build based on that revision. Because cloning a repository can be resource heavy, AnthillPro can store the cloned repository on the agent, so only the changes are updated on the cloned repository when the workflow is run again. To keep a cloned repository on the agent, (a.) *do not* add the Git Cleanup (remove it if you used the Job Wizard); and (b.) *do not* check the Clean Workspace box when configuring the Git Populate Workspace step. Otherwise, AnthillPro will clone the repository every time a build is run.

See Git Source Configuration.

Git Prerequisites

- Git must already be installed. See Git Documentation [<http://git-scm.com/documentation>].
- You must have administrative permissions. See Manage Security.
- The location of the `git` executable, if not in the system path, must be configured under the **System > Repositories**.

Set Up Git Repository

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **Git** from the drop-down menu and click **Set**.
4. Configure repository:
 - **Name** the repository. This is the name AnthillPro will use to identify this repository.
 - **Description.** Give an optional description.
 - **Command Path.** Provide the location of the `git` executable, if not in the system path. See Git Documentation

[<http://git-scm.com/documentation>] for information on identifying the Git executable.

5. Click **Set**.
6. Click **Done**.
7. See Git Source Configuration.

Git Source Configuration

Once the main Git repository is identified (see Set Up Git Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the Git repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Edit Source link on the Workflow Main page.)

1. Once Git is set up with AnthillPro (see Set Up Git Repository), create a project.
2. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see Create a New Workflow before continuing. Configure the source on the workflow:

- **Working Directory Script.** Select the Working Directory Script from the drop-down menu. The working directory is the location where the agent is going to run its commands. For example, if the working directory is `C:\projects\Project_A\` every command AnthillPro issues will be from that specific directory.

If no Working Directory Script is configured in the job, the script selected here will be used. See Working Directory Scripts.

- **Repository URL.** Give the location of the repository the workflow is to use. By default, AnthillPro clones this repository on the agent, and then uses the clone to build from. If, during job configuration, you add the Git Cleanup step (or check the Clean Workspace box when configuring the Git Populate Workspace step), the cloned repository will be removed every time a build is run. If you want to keep the cloned repository on the agent, see AnthillPro and Cloned Git Repositories.
- **Repository Name.** Give the name used to identify the repository in AnthillPro. This name does not correspond to the actual Git repository, and is simply an identifying label used by the AnthillPro system.
- **Revision.** Optionally, give the revision or tag name to use when cloning and/or updating the repository.
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/` -
- for any number of any characters except \ and /: `*` -
- for any single character / including \ and / Changelog file paths containing drive letter or any prefixes: `?` -

Make sure to include the `*` at the beginning of the path.

Example exclude paths: `-**/abc/**/*` Except for: `+**/abc/*.html`. This will exclude everything under the abc directory except for the HTML files in it.

3. Click **Save**.

Harvest

Information for connecting to a Harvest Repository can be shared by many projects. A new Harvest repository can be registered with AnthillPro in the Repositories section on the Administration page.

Harvest Prerequisites

- Harvest must already be installed.
- You must have administrative permissions. See Manage Security.

Set Up a Harvest Repository

To configure the repository:

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **Harvest** from the drop-down menu and click **Set**.
4. Configure repository:
 - **Name** the repository. This is the name AnthillPro will use to identify this repository.
 - **Description**. Give an optional description.
 - **Broker**. Enter the Harvest broker to connect through.
 - **User Name**. Give the user name used to connect to AnthillPro. For test purposes, this could be the Harvest user name of a developer or Administrator on the project. In production, we recommend creating a user exclusively for AnthillPro for clear traceability, security management and easier transition as the team changes.
 - **Password**. Provide the password corresponding to the account specified in the Username field. Select either: No Password, Use Password, or Use Script.
 - **No Password**. If no password is associated with AnthillPro's Harvest account, use this option.
 - **Use Password**. If a password is associate with AnthillPro's Harvest account, select this potion. Give the password in the Password field and confirm the password.
 - **Use Script (advanced)**. Give a parameterizable value that will resolve to the password. Use this instead of the Password field if the password will be stored in a secure property resolved at runtime. See Scripting.
 - **Revision Date Format**. Specify the format. AnthillPro retrieves revision histories for you project. Depending on your locale, it may format dates in different ways. If the default value fails for you, adjust this value to match the dates logged by the get changelog step of a build.
 - **Command Location**. If the Harvest command line client is not on the path, tell AnthillPro where it is installed on the agent(s).

5. Click **Set**.
6. Select the **Security** tab and click the **Edit** button. Determine **permissions**, click **Save**, and then click **Done**.
7. Click **Done**.

Harvest Source Configuration

The Repository configuration above declares how projects can connect to a Harvest Repository. A project needs to specify some rules for what source control should be retrieved to build from. Commonly, as a project matures, there will be a handful of source configurations created for the single project. Perhaps one for an old maintenance branch and another for the current development.

Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see [Create a New Workflow](#) before continuing. Configure the source on the workflow:

1. **Configuration Name and Description.** These are used exclusively as identifies within AnthillPro. Typical source configuration names would be 'Work State' to indicate the rules used. The description is optional and may go into more depth.
2. **Work Directory Script.** This determines where on the agent the project will be checked out and built from. The default value works well. It works from the a directory named after the project under the Agent's AnthillPro jobs directory.
3. **User Exclusions.** This feature is usually not used and can be ignored safely. To use it, list users whose changes should be ignored by the system. This is most commonly used when there is a special user that automatically updates the project on a regular basis whose changes are outside the scope of what AnthillPro should be interested in. Less frequently this feature is used to ignore users with access to the source who are not part of the core development team (documentation writers).
4. **File Exclusions.** This feature is usually not used and can be ignored safely. To use it, list file paths which AnthillPro should not track changes for. Most commonly this is used to exclude a 'docs' directory or some similar set or resource which the development team chooses to ignore from a process stand point.
5. **Project Name.** Enter the Harvest project name the sources should be retrieved from.
6. **State.** Enter the state from which to retrieve sources. If this source configuration is here to enable builds from a snapshot, the Snapshot state would be appropriate. Otherwise, a Work state is more likely.
7. **View Path.** Enter the view path to retrieve the sources from.
8. **Package.** Enter a package name here only if you want to restrict the retrieval of files to a particular package. To require a user to specify the package at build time, enter a scripted package name like `${property:PACKAGE}` and add a required PACKAGE property to the build from package workflow.
9. **File Pattern.** Enter the file pattern used to retrieve files from the view path. To retrieve every file on that path, use `"*"`.
- 10 **Package.** Enter a name here only if you want to restrict the retrieval of files to a particular package. To require a user to specify the snapshot at build time, enter a scripted snapshot name like `${property:Snapshot}` and add a required 'Snapshot' property to the build from snapshot workflow.

Harvest Steps

Every source control system has some similar actions that are used in the generic build jobs. But custom jobs can ex-

pose special behavior unique to the SCM tool. Steps are available to checkout source; to get a change log; to approve a package; to promote the code used in the build; to demote that code; and to create a new snapshot. Most of the steps (available under the Source Steps folder during job configuration) have minimal configuration but provide the option to specify the name of the Harvest process to use. For instance, if the 'checkout for browsing' process should be used instead of the default checkout process, that can be specified.

(Integrity) MKS Source Integrity

The first step in using an MKS Integrity repository with AnthillPro is to configure it from the Repositories item on the System page. This configuration will allow basic information about where the Integrity server is, and how to connect to it to be reused by several projects.

- **Name** the repository. This is the name AnthillPro will use to identify this repository.
- **Description.** Give an optional description.
- **Hostname.** The host name or IP address of the MKS Source Integrity server.
- **Port.** The port the Source Integrity server is listening on.
- **Username.** The MKS Source Integrity user name that will be used to connect to the server.
- **Password.** Provide the password corresponding to the account specified in the Username field. Select either: Set Password or Use Script.
 - **Set Password.** If a password is associate with AnthillPro's Integrity account, select this potion. Give the password in the Password field and confirm the password.
 - **Use Script (advanced).** Give a parameterizable value that will resolve to the password. Use this instead of the Password field if the password will be stored in a secure property resolved at runtime. See Scripting.
- **Command Path.** The location of the `si` executable. This value can take on a value that changes depending on the agent like, `${env\MKS_HOME}\bin`.
- **Date Format.** Give the date format to use when parsing change logs if other than the default MMM dd, yyyy - hh:mm a.

MKS Source Integrity Source Configuration

Each project will have on or more source configurations. These provide a template for a set of code to checkout. Typically, a source configuration is set up for each major concurrent line of development (branch). There may also be a template for building from a specific label.

Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see [Create a New Workflow](#) before continuing. Configure the source on the workflow:

- **Project File Name.** The project file AnthillPro should reference when communicating with MKS. By default that project file is named `project.pj` and that what AnthillPro assumes if left blank.
- **Project Directory.** The path to the project file on the MKS integrity server based on its directory structure.
- **Label Name.** Build from a specific label. This could be a hard-coded floating label. Or a dynamic label like `${property:label}` passed in at build time.
- **Branch Name.** Build from a branch. Like label, a dynamic property is allowed.

Mercurial

The first step in using a Mercurial repository with AnthillPro is to configure it by following the **Repositories** link on the **System** page. The configuration will allow basic information regarding Mercurial to be reused by several project workflows. Once configured, the repository will be listed on the **Repositories** main page.

Once the main repository has been identified, your workflows can then use the Mercurial repository. During project creation, associate the project workflow with a specific Mercurial repository. See **Mercurial Source Configuration**.

Mercurial Prerequisites

- Mercurial must already be installed. See **Mercurial Documentation** [<http://www.selenic.com/mercurial/wiki/index.cgi/Mercurial>].
- You must have administrative permissions. See **Manage Security**.
- The location of the `hg` executable, if not in system path, must be available.

Set Up Mercurial Repository

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **Mercurial** from the drop-down menu and click **Set**.
4. Configure repository:
 - **Name** the repository.
 - **Description**. Give an optional description.
 - **Command Path**. Provide the location of the `hg` executable, if not in the system path. See **Mercurial Documentation** [<http://www.selenic.com/mercurial/wiki/index.cgi/Mercurial>] for information on identifying the Mercurial executable.
5. Click **Set**.
6. Click **Done**.

Mercurial Source Configuration

Once the main Mercurial repository is identified (see **Set Up Mercurial Repository**), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the Mercurial repository. Once this is done, the **Source Configuration** page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the **Edit Source** link on the **Workflow Main** page.)

1. Once Mercurial is set up with AnthillPro (see **Set Up Mercurial Repository**), create a project.
2. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see **Create a New Workflow** before continuing. Configure the source on the workflow:

- **Working Directory Script.** Select the Working Directory Script from the drop-down menu. The working directory is the location where the agent is going to run its commands. For example, if the working directory is `C:\projects\Project_A\` every command AnthillPro issues will be from that specific directory.

If no Working Directory Script is configured in the job, the script selected here will be used. See Working Directory Scripts.

- **Repository URL.** Give the location of the repository the project is to use.
- **Repository Name.** Give the name of the existing repository. If one does not already exist, provide a name to create a new repository. This is the name AnthillPro will use to identify this repository.
- **Revision.** Optionally, give the revision or tag name to use when cloning and/or updating the repository.
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/` -
- for any number of any characters except `\` and `/:` `*` -
- for any single character / including `\` and / Changelog file paths containing drive letter or any prefixes: `?` -

Make sure to include the `*` at the beginning of the path.

Example exclude paths: `-**/abc/**/*` Except for: `+**/abc/*.html`. This will exclude everything under the abc directory except for the HTML files in it.

3. Click **Save**.

Perforce

The first step in using a Perforce repository for your AnthillPro projects is to let AnthillPro know where your repository is, and how to connect to it. This is done on the System page, so you will need administrative permissions to configure a repository. Once the repository configuration is set up, each workflow will need to be configured with the project's specific source configuration. To get started:

1. Go to **System > Repositories > Create New**.
2. Select **Perforce** from the drop-down menu and click **Set**.

Depending on how AnthillPro was installed, you may see Perforce-Plugin in the menu. While the Plugin integration differs, the general configuration process is the same.

- The Perforce Plugin allows you to past a copy of the Perforce client spec during source configuration if you set AnthillPro to create a new client spec -- this option is not available in the standard integration.

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbancode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

The remainder of this section assumes you are using the standard integration.

3. Configure the repository:

- **Name.** Give a name for this configuration. What you give here will be used throughout AnthillPro. For example, when performing source configuration on a workflow, the name given here will be displayed to the user.
- **Description.** You can give a short description, if desired. Often, users will put the repo's URL in the description, especially if they have multiple repositories configured.
- **Port.** This is the server name and port used to connect to the Perforce server. Perforce defaults to the 1666 port.
- **Username.** The user name that AnthillPro will use to connect to the server. You can either create a new Perforce user (e.g., Anthill) or have AnthillPro connect to Perforce via an existing user.
- **Password.** Provide the password corresponding to the account specified in the Username field. Select either: No Password, Use Password, or Use Script.
 - **No Password.** If no password is associated with AnthillPro's Perforce account, use this option.
 - **Use Password.** If a password is associate with AnthillPro's Perforce account, select this potion. Give the password in the Password field and confirm the password.
 - **Use Script (advanced).** Give a parameterizable value that will resolve to the password. Use this instead of the Password field if the password will be stored in a secure property resolved at runtime. See Scripting.
- **Command Path.** Provide the location of the Perforce executable (p4) if it is not in the system path. This value can take on a value that changes depending on the agent, for example `${env\P4_HOME}`.
- **Character Set.** The optional client character set.
- **Command Character Set.** The optional client command-line character set.

4. Click **Save**. To use Perforce with an AnthillPro project, see Perforce Source Configuration.

Perforce Source Configuration

To configure the source for a project, make sure you have already configured your Perforce repository in AnthillPro. Each project may have one or more source configurations that provide a template for a set of code AnthillPro will checkout. Typically, a source configuration is set up for each major concurrent line of development (branch or view). There may also be a template for building from a label.

If you are using the Perforce Plugin, you can past a copy of the Perforce client spec during source configuration if you set AnthillPro to create a new client spec.

Once you have created a project, you will need to select the **add workflow** icon under the Actions menu. Give the workflow's basic configuration and then configure the source as follows:

1. Select the Perforce repository configuration you created earlier and click **Set**. See Perforce if you have not configured the repository yet.
2. Go to the originating workflow that was created as part of the project creation process. If you have not created an

originating workflow yet, see [Create a New Workflow](#) before continuing. Configure the source on the workflow:

- **Client.** There are two options for setting up the Perforce client. Use either an existing Perforce client configured on the build machine(s) or create a new client spec at runtime.
 - **Use existing client spec.** Give the name of the Perforce client that should be used. If you only have a single build machine, this option should be fine. However, when multiple machines (i.e., agents) are used to run a build, this approach can introduce risk as the configuration of the clients may not be done at the same time. If you have multiple build machines, it is best to use the **Create new client spec** option.
 - **Create new client spec.** Give a template for a new client, and have AnthillPro set up the clients on the fly with each build. The Working Directory Script dictates where the client will be put on the host machine.

Name. This name corresponds with Perforce client names. For the name of the new client spec, you can use a script that would ensure a unique name for each agent in order to avoid problems with concurrent builds. You can use something similar to `myproject-${agent:locked/agent.name}` and replace 'myproject' with the correct project name.

Template Name. Give the name of the Perforce template you want AnthillPro to use to create the client.

Working Directory Script. Select the Working Directory Script from the drop-down menu. The working directory is the location where the agent is going to run its commands. For example, if the working directory is `C:\projects\Project_A` every command AnthillPro issues will be from that specific directory. If no Working Directory Script is configured in the job, the script selected here will be used. See [Working Directory Scripts](#).

- **Label Script.** Build from a specific label. This could be a hard-coded floating label. Or a dynamic label like `${property:label}` passed in at build time.
- **Delete Workspace Files.** Select to do clean builds (deletes all the files in the workspace before repopulating the workspace).
- **Do Not Update the Perforce Have List.** Enable this to pass the '-p' option on the p4 sync command that will not update the Perforce have list. This option is only available to later versions of Perforce. Using this at the source config level will prevent you from being able to label the source checked out. You can also set this on the individual populate workspace step.
- **Preserve Unlabeled Files.** This is used for patch builds. If you are checking out from a label (see above) and want to check the label out over the top of an existing workspace.
- **Do Not Limit Labels to Client Spec.** This is used to apply the same label to multiple projects. This means that a label will be applied cumulatively.
- **Login Before Each Step.** If Perforce authentication is done through LDAP, commands must be run after doing a full login, not just passing the credentials at the command line. This check box will use that approach.
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/` -
- for any number of any characters except \ and /: `*` -
- for any single character / including \ and / Changelog file paths containing drive letter or any prefixes: `?` -

Make sure to include the * at the beginning of the path.

Example exclude paths: -**/abc/**/* Except for: +**/abc/*.html. This will exclude everything under the abc directory except for the HTML files in it.

3. Click **Save**.

PVCS Source Control

To use a PVCS repository with AnthillPro, configure it from the Repositories item on the System page. The configuration will allow basic information regarding the PVCS server to be reused by several project workflows. Once configured, the repository will be listed on the Repositories main page.

See also PVCS Tracker.

PVCS Prerequisites

- You must have administrative permissions. See Manage Security.
- The PVCS Project Database path must be available to complete the configuration.

Set Up PVCS Repository

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **PVCS** from the drop-down menu and click **Set**.
4. Configure repository:
 - **Name** the repository.
 - **Description.** Give an optional description.
 - **Username.** Give the name AnthillPro should use to connect to the repository.
 - **Password.** Provide the password corresponding to the account specified in the Username field. Select either: No Password, Use Password, or Use Script.
 - **No Password.** If no password is associated with AnthillPro's PVCS account, use this option.
 - **Use Password.** If a password is associate with AnthillPro's PVCS account, select this potion. Give the password in the Password field and confirm the password.
 - **Use Script (advanced).** Give a parameterizable value that will resolve to the password. Use this instead of the Password field if the password will be stored in a secure property resolved at runtime. See Scripting.
 - **Project Database.** Provide the path to the project database (variables may be used).
 - **Command Line Date Format.** Give the date format to use with vlog command.

- **Changelog Date Format.** Provide the date format used in reports that were created using the `vlog` command.
 - **Command Path.** Provide the path to the `pcli`, `vcs`, and `vlog` executables if they are not in the system path.
5. Click **Set**.
 6. Select the **Security** tab. See Manage Security.
 7. Click **Edit**, determine **permissions**, click **Save**, and then click **Done**.
 8. Click **Done**.

Rational Team Concert (SCM)

The first step in using a Rational Team Concert (RTC) repository with AnthillPro is to configure it by following the Repositories link on the System page. The configuration will allow basic information regarding RTC to be reused by several project workflows. Once configured, the repository will be listed on the Repositories main page.

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbanocode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

After the main repository has been identified, your workflows can then use the RTC repository. During project creation, associate the project workflow with a specific RTC repository (see Rational Team Concert (SCM) Source Configuration) and create a build job that includes the RTC job steps.

Rational Team Concert (SCM) Prerequisites

- You must have administrative permissions. See Manage Security.
- The RTC client must be installed on your AnthillPro agents in the build environment. See Environments for more.
- The RTC server URL is required to configure the integration.
- The RTC user name and password AnthillPro will use to contact the server must be created in RTC prior to configuring the integration.

Set Up Rational Team Concert (SCM) Repository

The RTC integration must be configured before you can run any builds. The information given here will be reused by all of your projects that use the RTC repository. If you have multiple instances of RTC (i.e., they are completely different URLs), configure an integration for each one.

1. Go to **System > Repositories** under the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **RTC-SCM-Plugin** from the drop-down menu and click **Set**.
4. On the **Main** tab, configure repository:

- **Name** the repository.
- **Description.** Give an optional description.
- **Command Path.** Provide the full path of the SCM executable, including the file name. If the executable is already on the path, leave this field blank.
- **RTC server URL.** Enter the URL of the RTC server. For example: `https://myserver:9443/jazz`.
- **Repository Username.** Give the user name to use when accessing the repository.
- **Repository Password.** Provide the password corresponding to the account specified in the Repository Username field.
 - Confirm password.

5. Click **Done** and see Rational Team Concert (SCM) Source Configuration.

Rational Team Concert (SCM) Source Configuration

Once the main RTC repository is identified (see Set Up Rational Team Concert (SCM) Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the RTC repository type. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Edit Source link on the Workflow Main page.)

1. Once RTC is set up with AnthillPro, create a project. If you are unfamiliar with AnthillPro, see Setting Up a Build for help.
2. Go to the originating workflow. If you have not created an originating workflow yet, see Create a New Workflow before continuing. Configure the source on the workflow:

- **Working Directory Script.** Select the Working Directory Script from the drop-down menu. The working directory is the location where the agent is going to run its commands. For example, if the working directory is `C:\projects\Project_A` every command AnthillPro issues will be from that specific directory.

If no Working Directory Script is configured in the job, the script selected here will be used. See Working Directory Scripts.

- **Name.** Give a name for this configuration to be displayed on the Main tab.
- **Workspace Name.** Enter the name of the workspace that is going to be used. If the workspace does not exist one will be created based on the name you give here. You can use an alias as well but only for existing workspaces.
- **Stream.** Provide the name or alias of the default flow target stream for the workspace above.
- **Directory Offset.** Give the offset from the current working directory where the local workspace is created. If there is no offset, leave this field blank.
- **Build Snapshot Name.** Specify the name of the snapshot AnthillPro should create for every build. For example, if you use the default of `"anthill-{bsh:BuildLifeLookup.getCurrent().getId()},"` the snapshot created will append "anthill-" with the current Build Life.
- **Build Snapshot Description.** Provide the description of the snapshot AnthillPro should create for every build. This is the description of the snapshot given above. For example, if you use the default of "Created by Ant-

hillPro for Buildlife: `{bsh:BuildLifeLookup.getCurrent().getId() }`, " the description created will append "Created by AnthillPro for Buildlife:" with the current Build Life.

- **Component List.** Give a comma-separated list of components (name or alias) to load in the local workspace. Leave blank to load all components.
- **Include Root.** Select here to load component roots as directories on the file system.
- **Force.** Select here to overwrite existing files when loading.
- **Baseline.** If you want to build from a baseline, enter the baseline name or alias here. If you specify a snapshot below, leave this field blank.
- **Snapshot.** If you want to build from a snapshot, enter the snapshot name or alias here. If you specify a snapshot here, do not define the baseline above.
- **Exclude Users.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File Filters.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/` -
- for any number of any characters except `\` and `/`: `*` -
- for any single character / including `\` and / Changelog file paths containing drive letter or any prefixes: `?` -

Make sure to include the `*` at the beginning of the path.

Example exclude paths: `-**/abc/**/*` Except for: `+**/abc/*.html`. This will exclude everything under the abc directory except for the HTML files in it.

- **Repository.** Select the correct integration from the drop-down menu (this is the integration configured in the Set Up Rational Team Concert (SCM) Repository section). If you configured multiple RTC repositories on the AnthillPro System page, make sure you select the correct one.

3. Click **Save** and see Rational Team Concert (SCM) Build Jobs.

Rational Team Concert (SCM) Build Jobs

Typically, your RTC build job will include the following steps, in the order presented (the Job Wizard can be a helpful tool when configuring the job):

1. **Cleanup Workspace.** This step cleans up the workspace used for previous builds.
2. **Populate Workspace.** Populates the workspace with the files necessary to run the build.
3. **Get Changelog.** This step works by locating a prior Build Life and getting the source changes since the prior Build Life up to the current source. The prior Build Life is located by using status and/or stamp requirements.
4. **Create Stamp.** Applies a unique stamp to the build.
5. **Resolve Dependency Artifacts.** Retrieves all artifacts generated this Build Life's dependencies.

6. **Build.** This step will execute your build script (e.g., Ant, Groovy, Make) or invoke another builder.
7. **Label Source.** Allows you to specify a particular label that you would like to apply to the source.
8. **Publisher.** The publisher steps typically send information and files associated with the build to other parts of the AnthillPro system or AnthillPro users via notifications. For example, the Artifact Deliver step that sends the build artifacts to Codestation, AnthillPro's artifact-management system.
 - There are a number of other publishers you can include. You can view them on the Steps page in the Publishers folder.
9. **Assign Status - Success.** Assigns the status of Success if the job completes.
- 10 **Assign Status - Failure.** Assigns the status of Failure if the job does not complete.

In addition, AnthillPro has four RTC-specific steps you can use. Typically, these steps will not be used; however, depending on your processes, it may be necessary to use one of the RTC-specific steps. It is recommended to configure your build job as outlined above. This will allow you to easily switch source, etc.:

- **RTC-SCM Cleanup.** Perform a cleanup of the current working directory.
- **RTC Changelog.** Perform a RTC changelog. By default the retrieved changelog will be between the previous and current Build Life.
- **RTC Create Snapshot.** Snapshots are created as part of the Populate Workspace/Get Source step, this step should not be used.
- **RTC Get Source.** Perform the necessary steps to acquire source from RTC as indicated by source configuration.

StarTeam

The first step in using an StarTeam repository with AnthillPro is to configure it from the Repositories item on the System page. This configuration will allow basic information about where the StarTeam server is, and how to connect to it to be reused by several projects.

- **Server.** The host name or IP address of the StarTeam server. You can find this on the server settings dialog in the StarTeam Client.
- **Port.** The port the StarTeam server is listening on. You can find this on the server settings dialog in the StarTeam Client.
- **Username.** The StarTeam user name that will be used to connect to the server.
- **Password.** Provide the password corresponding to the account specified in the Username field. Select either: No Password, Use Password, Use Script, or Use File.
 - **No Password.** If no password is associated with AnthillPro's StarTeam account, use this option.
 - **Use Password.** If a password is associate with AnthillPro's StarTeam account, select this potion. Give the password in the Password field and confirm the password.
 - **Use Script (advanced).** Give a parameterizable value that will resolve to the password. Use this instead of the Password field if the password will be stored in a secure property resolved at runtime. See Scripting.

- **Use File.** Instead of using the password field above, if this is set it will use the password found in the specified file.
- **Date Format.** StarTeam dates that must be parsed by AnthillPro can change based on Locale. If you are not using US/English date format, change this value.
- **Jar File Location.** This is the full path to the StarTeam jar library AnthillPro uses to retrieve change logs. The full path including the starteamXX.jar file name is required. This is usually found in a StarTeam SDK folder.
- **Command Path.** The location of the **stcmd** executable. This value can take on a value that changes depending on the agent like `${env\STAR_HOME}\bin`.

StarTeam Source Configuration

Each project will have on or more source configurations. These provide a template for a set of code to checkout. Typically, a source configuration is set up for each major concurrent line of development (branch or view). There may also be a template for building from a specific label.

Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see [Create a New Workflow](#) before continuing. Configure the source on the workflow:

- **Project Name.** The name of the StarTeam project. A project is the container for one or more views.
- **View.** If AnthillPro is building from a view other than the default, name that view here.
- **Path.** If only a subset of the view should be retrieved and built from, provide that path.
- **Label Script.** Build from a specific label. This could be a hard-coded floating label. Or a dynamic label like `${property:label}` passed in at build time.

Subversion (SVN)

The first step in using a Subversion repository with AnthillPro is to configure it by following the [Repositories](#) link on the [System](#) page. The configuration will allow basic information regarding Subversion to be reused by several project workflows. Once configured, the repository will be listed on the [Repositories](#) main page.

Once the main repository has been identified, your workflows can then use the Subversion repository. During project creation, associate the project with a specific Subversion repository. See [Subversion Source Configuration](#).

Subversion Prerequisites

- You must have administrative permissions. See [Manage Security](#).
- The Subversion Root URL must be available to complete the configuration.
- If using the ViewVC integration, see [ViewVC](#).

Set Up Subversion Repository

1. Go to **System > Repositories** under the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **Subversion** from the drop-down menu and click **Set**.
4. On the **Main** tab, configure repository:

- **Name** the repository.
- **Description.** Give an optional description.
- **Root URL.** Provide the base URL of the repository.
- **Username.** Give the user name to use when accessing the repository.

If using the ViewVC integration, the Username field must be the ViewVC repository name. It is not possible to use ViewVC if properties are set for the Username field. See ViewVC.

- **Password.** Provide the password corresponding to the account specified in the Username field. Select either: No Password, Use Password, or Use Script.
 - **No Password.** If no password is associated with AnthillPro's Subversion account, use this option.
 - **Use Password.** If a password is associate with AnthillPro's Subversion account, select this potion. Give the password in the Password field and confirm the password.
 - **Use Script (advanced).** Give a parameterizable value that will resolve to the password. Use this instead of the Password field if the password will be stored in a secure property resolved at runtime. See Scripting.
 - **Command Path.** Provide the location of the *svn* executable, if not in system path.
 - **Source Viewer.** Select the Source Viewer Type from the drop-down menu (if using the ViewVC integration). Otherwise select **None**. See ViewVC.
5. If not setting a **Repository Trigger** or **Security**, click **Set** then **Done** to complete. Otherwise proceed to **item 6** to set a trigger or **item 8** to set security.
 6. Select the **Trigger** tab. To either **deactivate** or **delete** the repo trigger, click the appropriate button. If the repo trigger is deactivated/deleted, you will need to configure a workflow trigger for every project. See Using Triggers with Subversion.
 7. If not setting **Security**, click **Activate** then **Done** to complete. Otherwise click **Activate** and proceed to **item 8**.
 8. Select the **Security** tab and click the **Edit** button. Check the appropriate boxes to determine user-role access (See Manage Security), and click **Save**.
 9. Click **Done**.

Subversion Source Configuration

Once the main Subversion repository is identified (see Set Up Subversion Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the Subversion repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Edit Source link on the Workflow Main page.)

See also Using Triggers with Subversion.

1. Once Subversion is set up with AnthillPro (see Set Up Subversion Repository), create a project. See Setting Up a Build.
2. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see Create a New Workflow before continuing. Configure the source on the workflow:

- **Working Directory Script.** Select the Working Directory Script from the drop-down menu. The working directory is the location where the agent is going to run its commands. For example, if the working directory is `C:\projects\Project_A\` every command AnthillPro issues will be from that specific directory.

If no Working Directory Script is configured in the job, the script selected here will be used. See Working Directory Scripts.

- **Directory Offset.** Give the sub-directory where this module is to be placed within the working directory. If the root of the working directory is to be used, leave this item blank.
- **Source URL.** Provide the URL of the project source. Note that *the Source URL is relative to the selected repository root URL*.
- **Tag.** If using tags, give the tag to use for checkout. This may include variables passed to the workflow via `${property:Name}` format. See Scripting.
- **Revision.** Give the Revision to use for checkout (if any). This may include variables passed to the workflow via `${property:Name}` format. See Scripting and Using Triggers with Subversion.
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/ -`
- for any number of any characters except \ and /: `* -`
- for any single character / including \ and / Changelog file paths containing drive letter or any prefixes: `? -`

Make sure to include the `*` at the beginning of the path.

Example exclude paths: `-**/abc/**/*` Except for: `+**/abc/*.html`. This will exclude everything under the abc directory except for the HTML files in it.

3. Click **Save**.

See also Using Triggers with Subversion.

Using Triggers with Subversion

Subversion (SVN) repository triggers allow you to create a single hook in the Subversion repository that is capable of triggering every workflow using this repository. In the post-commit hook script in your SVN repository, add a check to see that the changed information is on the path to the project. Then use a utility like Wget to generate an HTTP request to the URL specified in the trigger.

By default, AnthillPro activates the Subversion repository trigger when the repository is configured. When configuring an AnthillPro project, the repo trigger is automatically generated for every workflow. Under some conditions, it may be necessary to deactivate the repository trigger for individual workflows. See Ignoring Subversion Repository

Trigger.

If using **Windows**, see also Subversion Post-commit Hook and Windows.

Using Wget to Generate HTTP Request

Use a utility such as Wget to generate an HTTP request to the URL specified in the trigger. To build from trunk, it should look something like the following:

```
if [ -n "$(svnlook changed /data/svn/mycompany/myproject -r $2 | grep
trunk)" ]
then
    wget -t 0 https://server/trigger --post-data='triggerId=31&code
=95cde532437151c551cf062bf93c0d12de9209c7'
fi
```

If building from a branch, change the script to something like:

```
if [ -n "$(svnlook changed /data/svn/mycompany/myproject -r $2 | grep
branches/june_release)" ]
then
    wget -t 0 https://server/trigger --post-data='triggerId=31&code
=cf062bf93c0d12d95cde532437151c551e123456'
fi
```

To build based on the Subversion revision ID, change the script to:

```
post-data="code=$CODE&dirs-changed=$DIRSCHANGED&revision-date=$REVISION_DATE&revision-id=$
```

- Additionally, on the workflow source configuration, you will also need to change the **Revision** field to something similar to the following:

```
${bsh:PropertyLookup.get("revision-id") == null ? "" :
PropertyLookup.get("revision-id")}
```

Subversion Post-commit Hook and Windows

In a Windows environment, configuration of the Subversion post-commit hook requires a Wget Windows port [<http://users.ugent.be/~bpuype/wget/>] (due to Windows shell command restrictions). Subversion hook scripts use the absolute path to the programs because they call the clean Windows PATH variable for the entire environment.

In order to complete the configuration:

1. Download the Wget Windows port [<http://users.ugent.be/~bpuype/wget/>].
2. Place the **Wget Windows port** in the **hooks directory**.
3. **Name** the script **script.vbs** (it can be written as VBScript) and place it in the **hooks directory**.

Example Subversion Post-commit Hooks Script for Windows:

```
TRIGGER_URL = "https://localhost:8443/trigger"
REPOS = WScript.Arguments.Item(0)
REV = WScript.Arguments.Item(1)
CODE = "c6ae3e9eeaebb6830aca6e7fb51247b39d410ab3"
```

```
dq = chr(34)

Set oWSH = CreateObject( "WScript.Shell" )
Set nRet = oWSH.Exec( "C:\your\path\to\Subversion\bin\svnlook.exe _
  dirs-changed " & REPOS & " -r " & REV)
DIRSCHANGED = nRet.StdOut.ReadAll

Set nRet2 = oWSH.Exec( "C:\your\path\to\hooks\wget.exe --tries 1 _
  --no-check-certificate " & dq & TRIGGER_URL & dq & " --post-data=" & dq & _
  "code=" & CODE & "&dirs-changed=" & DIRSCHANGED & dq )
```

4. Place a `post-commit.bat` in the `hooks` directory.

- Because Subversion is unable to natively call the VBScript, the post-commit hook must be invoked from either a `.bat` or `.exe` file.

Example `post-commit.bat` script:

```
C:\your\path\to\repository\hooks\script.vbs %1 %2
```

Ignoring Subversion Repository Trigger

By default, AnthillPro activates the Subversion repository trigger when the repository is configured. When configuring an AnthillPro project, the repo trigger is automatically created and added to every workflow. Under some conditions, it may be necessary to deactivate the repository trigger for individual workflows: It is common for a project to have multiple workflows that must build independently of each other, and in this case the repository trigger may kick off a workflow unnecessarily. For example, a project might have 3 workflows that need to build independently of each other: Workflow A just compiles code; workflow B performs nightly build; and workflow C builds a release candidate. To keep the workflows, which all use the same repository, independently building (i.e., build when you want them to) disable the repository trigger *on the workflow* and create a workflow trigger (see Use Triggers and Scheduled Builds).

1. Go to **Administration**, select the appropriate workflow, and select the **Triggers** tab. If you can't access this page, that means you don't have the appropriate permissions. Contact your AnthillPro administrator.
2. Click the **Ignore Repository Trigger** icon under the Actions menu.

Once the trigger is disabled, the icon will change from green to brown. To enable the trigger for this workflow, simply click the icon again.

3. Create a workflow trigger. See Use Triggers and Scheduled Builds.
4. Repeat Items 1 thru 3 for every workflow that will not use the repository trigger.

Building Multiple Branches of Source with the SVN Repository Trigger

If you need to build multiple branches of source within the same workflow, you will need to modify SVN Get Changelog step in your build job. This will allow any properties that are passed on the trigger to determine what is built based on the Start Stamp Pattern. Your Stamp should contain the source branch in it. In the example below, the property is named "source":

```
build-${property:source}-${bsh:BuildLifeLookup.getCurrent().getId() }
```

Synergy

To use a Synergy repository with AnthillPro, configure it from the Repositories item on the Administration page. The configuration will allow basic information regarding the Synergy repo to be reused by several project workflows. Once configured, the repository will be listed on the Repositories main page and available during source configuration when creating a project.

Synergy Prerequisites

- You must have administrative permissions for the System page. See Manage Security.
- The Synergy database path must be available to complete the configuration.
- AnthillPro must be set up as a Synergy user.

Set Up Synergy Repository

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **Synergy** from the drop-down menu and click **Set**.
4. Configure repository.
 - **Name** the repository.
 - **Description (optional)**. Provide a description
 - **Username**. Enter the user name to use when accessing the repository.
 - **Password**. Provide the password corresponding to the account specified in the Username field. Select either: No Password, Use Password, or Use Script.
 - **No Password**. If no password is associated with AnthillPro's Synergy account, use this option.
 - **Use Password**. If a password is associate with AnthillPro's Synergy account, select this potion. Give the password in the Password field and confirm the password.
 - **Use Script (advanced)**. Give a parameterizable value that will resolve to the password. Use this instead of the Password field if the password will be stored in a secure property resolved at runtime. See Scripting.
 - **Database path**. Enter the path to the Synergy Database (may be a UNC path).
 - **Home Directory**. When starting a session, Synergy writes information to the home directory of the user the command is running as. If the user (the AnthillPro agent is running as) has no home directory, you can specify another directory for this information to be written to.
 - **Database Info Directory**. This option is only used when using the remote client: i.e., if you check the **Use Remote Client** box below. If you enter a directory here but do not use the remote client, AnthillPro will ignore this setting, and use the default /tmp/cmm directory.

When starting a session with a remote client, Synergy writes out certain database connection information to the file system. The default directory is /tmp/ccm. If you need it to write to a different directory, specify the directory here.

- **Role (optional).** Enter the role to login with for the provided user name. If no role is specified, AnthillPro will use "build_mgr". Only those with a "build_mgr" role can label and/or create a baseline.
 - **Host (optional).** Enter the host name of the machine Synergy is running on.
 - **Maintain Session.** Check the box to maintain a single Synergy Session per job. If a Synergy session is slow to establish, checking the box will ensure the job runs correctly.
 - **Use Remote Client.** Check the box to remove the client option when starting a Synergy session. If checked, AnthillPro will pass the `-rc` flag and remove the client when starting a Synergy session. If you check this box, you will need specify the **Database Info Directory** above.
 - **Command path.** Enter the location of the `ccm` executable (if not in the system path).
 - **Date Format.** Determine the date format to use when parsing change logs. The default (**EEE, MMM dd, yyyy hh:mm:ss a**) corresponds to the Java SimpleDateFormat that is output by the Synergy `get-changelog` command.
5. Click **Set**.
 6. Select the **Security** tab. See Manage Security.
 7. Click **Edit**, determine **permissions**, click **Save**, and then click **Done**.
 8. Click **Done**.

Synergy Source Configuration

Once the main Synergy repository is identified (see Set Up Synergy Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the Synergy repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Configure, or Edit Source, link under the Source Config menu on the Workflow Main page.)

1. Once Synergy is set up with AnthillPro (see Set Up Synergy Repository), create a project.
2. Select the appropriate Synergy repository from the drop-down menu and click **Set**.
3. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see Create a New Workflow before continuing. Configure the source on the workflow:
 - **Project Spec.** Give the Project Spec for the Synergy Project. The Project Spec is defined in Synergy, and is required here.
 - **Release.** Enter the Synergy Release to use when creating a baseline. If you want to perform a label step, you must give a release here.
 - **Purpose.** Enter the synergy purpose to use when creating a baseline. If nothing is given, default is: Integration Testing.
 - **Checkout Strategy.** Select a strategy. Either perform a checkout if one does not exist or use an existing checkout when dealing with Synergy Project Specs.
 - **Work Area.** Give the Work Area of the Synergy Project Spec to be used.
 - **Working Directory Script.** Select the Working Directory Script from the drop-down menu. The working dir-

ectory is the location where the agent is going to run its commands. For example, if the working directory is C:\projects\Project_A\ every command AnthillPro issues will be from that specific directory.

If no Working Directory Script is configured in the job, the script selected here will be used. See Working Directory Scripts.

- **Reconf.** Enter the value to use for a checkout.
- **Baseline.** Give the baseline to checkout from.
- **Versions.** Give the value to use for a checkout.
- **Checkout Purpose.** Enter the purpose for the checkout. For example, Collaborative Development and Insulated Development may be used.
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: **/ -
- for any number of any characters except \ and /: * -
- for any single character / including \ and / Changelog file paths containing drive letter or any prefixes: ? -

Make sure to include the * at the beginning of the path.

Example exclude paths: -**/abc/**/* Except for: +**/abc/*.html. This will exclude everything under the abc directory except for the HTML files in it.

4. Click **Save**.

Team Foundation Server (TFS) Source Control

The first step in using a TFS repository with AnthillPro is to configure it by following the Repositories link on the System page. The configuration will allow basic information regarding TFS to be reused by several project workflows. Once configured, the repository will be listed on the Repositories main page.

AnthillPro supports both the 2005 and 2008 versions of TFS. Repository set up and source configuration are the same for both.

Once the main repository has been identified, your workflows can then use the TFS repository during workflow source configuration. During project creation, the workflow is associated with a specific TFS repository.

AnthillPro may also be configured to create a TFS WorkItem, add comments, publish a WorkItem report, and Resolve WorkItems. See Team Foundation Server (TFS) Project Tracking.

TFS Prerequisites

- You must have administrative permissions. See Manage Security.

- The TFS URL must be available to complete the configuration.
- The TFS client must be installed on the same build machine as the agent so that AnthillPro can run TFS commands.

Set Up TFS Repository

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **TFS** from the drop-down menu and click **Set**.
4. Configure repository:
 - **Name** the repository. This is the name AnthillPro will use to identify this repository.
 - **Description**. Give an optional description.
 - **TFS Server URL**. Provide the repository URL.
 - **TFS Version**. From the drop-down menu, select the version of TFS you are using.
 - **Command Path**. Provide the absolute path to the *tf* executables if not in the system path.
5. Click **Set**.
6. Select the **Security** tab. See Manage Security.
7. Click **Edit**, determine **permissions**, click **Save**, and then click **Done**.
8. Click **Done**.
9. To set up a repository trigger, see Using Triggers with TFS.

TFS Source Configuration

Once the main TFS repository is identified (see Set Up TFS Repository), configure the specific repository a workflow uses. During project creation, the originating workflow is first associated with the TFS repository. Once this is done, the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Configure, or Edit Source, link under the Source Config menu on the Workflow Main page.)

1. Once TFS is set up with AnthillPro (see Set Up TFS Repository), create a project.
2. Select the appropriate TFS repository from the drop-down menu and click **Set**.
3. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see Create a New Workflow before continuing. Configure the source on the workflow:
 - **Repository**. Select the repository from the drop-down menu. By default, AnthillPro will use the repository selected in the previous step. To change TFS repositories, select a different repository from the drop-down menu. If only one repo is listed, then only one TFS repository is configured. If changing repositories, make sure to re-configure all remaining fields.

- **Working Directory Script.** Select the Working Directory Script from the drop-down menu. The working directory is the location where the agent is going to run its commands. For example, if the working directory is `C:\projects\Project_A` every command AnthillPro issues will be from that specific directory.

If no Working Directory Script is configured in the job, the script selected here will be used. See Working Directory Scripts.

- **Workspace Name.** Give the name of the TFS workspace to use. A property (via `${property:Name}` format) or short script may be used here.
- **Source URL.** Give the project-source URL, relative to the selected repository root URL. A property (via `${property:Name}` format) or short script may be used here. For example, you can script the SourcePath Module.
- **Tag.** If checking out from a tag, give it here. This may include variables passed to the workflow via `${property:Name}` format, or may be hard coded.
- **Revision.** Give the Revision to use for checkout. This may include variables passed to the workflow via `${property:Name}` format, or may be hard coded.
- **Directory Offset.** Give the directory to retrieve files to (i.e., checkout destination), if the TFS default directory structure is not desired. This may include variables passed to the workflow via `${property:Name}` format.
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/ -`
- for any number of any characters except `\` and `/`: `* -`
- for any single character / including `\` and `/` Changelog file paths containing drive letter or any prefixes: `? -`

Make sure to include the `*` at the beginning of the path.

Example exclude paths: `-**/abc/**/*` Except for: `+**/abc/*.html`. This will exclude everything under the abc directory except for the HTML files in it.

4. Click **Save**.

Add Additional Source with TFS

If adding additional sources to the workflow is necessary, set up is similar to that of the initial source configuration (see TFS Source Configuration). When adding additional sources, all URLs are relative to the root URL.

Once an additional source has been added to the workflow, it will appear on the workflow Main tab under the Source Config menu, and may be edited or deleted using the icons under the Actions menu.

To add an additional source to an existing workflow:

1. Go to **Administration** and select the workflow that an additional source is to be added to.

2. On the workflow **Main** page, select the **Add Additional Source** link under the Source Config menu.
3. Configure source:
 - **Source URL.** Give the project-source URL, relative to the selected repository root URL.
 - **Tag.** If checking out from a tag, give it here. This may include variables passed to the workflow via `${property:Name}` format, or may be hard coded.
 - **Revision.** Give the Revision to use for checkout. This may include variables passed to the workflow via `${property:Name}` format, or may be hard coded.
4. Click **Save**.

Using Triggers with TFS

TFS repository triggers allow you to create a single hook in the repository that is capable of triggering every workflow using this repository. By default, AnthillPro activates the TFS repository trigger when the repository is configured. When configuring an AnthillPro project, the repo trigger is automatically generated for every workflow. Under some conditions, it may be necessary to deactivate the repository trigger for individual workflows. See Ignoring TFS Repository Trigger.

1. Once the TFS repository is set up, click the **Trigger** tab.
2. Copy the TFS command that AnthillPro automatically generated.
3. Replace the `<serverName>` with the name of your TFS Server.
4. Go to your TFS Setup directory and execute the command. This will setup a SOAP notification to AnthillPro on check-in that contains information about changes.
5. You can either deactivate or delete the trigger. Only one repository trigger may be active at any one time. If you deactivate a trigger, you can return to it and reactivate it; if you delete a trigger, you can create a new one by repeating the previous steps.
6. See also Ignoring TFS Repository Trigger.

Ignoring TFS Repository Trigger

By default, AnthillPro activates the TFS repository trigger when the repository is configured. When configuring an AnthillPro project, the repo trigger is automatically created and added to every workflow. Under some conditions, it may be necessary to deactivate the repository trigger for individual workflows: It is common for a project to have multiple workflows that must build independently of each other, and in this case the repository trigger may kick off a workflow unnecessarily. For example, a project might have 3 workflows that need to build independently of each other: Workflow A just compiles code; workflow B performs nightly build; and workflow C builds a release candidate. To keep the workflows, which all use the same repository, independently building (i.e., build when you want them to) disable the repository trigger *on the workflow* and create a workflow trigger (see Use Triggers and Scheduled Builds).

1. Go to **Administration**, select the appropriate workflow, and select the **Triggers** tab. If you can't access this page, that means you don't have the appropriate permissions. Contact your AnthillPro administrator.
2. Click the **Ignore Repository Trigger** icon under the Actions menu.

Once the trigger is disabled, the icon will change from green to brown. To enable the trigger for this workflow, simply click the icon again.

3. Create a workflow trigger. See [Use Triggers and Scheduled Builds](#).
4. Repeat Items 1 thru 3 for every workflow that will not use the repository trigger.

Vault (SourceGear)

To use a SourceGear Vault repository with AnthillPro, configure it from the [Repositories](#) item on the Administration page. The configuration will allow basic information regarding the Vault server to be reused by several project workflows. Once configured, the repository will be listed on the [Repositories](#) main page.

Vault Prerequisites

- You must have administrative permissions. See [Manage Security](#).
- AnthillPro must be set up as a Vault user.
- The command path of the executable and the host name must be available.

Vault Configuration

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **Vault** from the drop-down menu and click **Set**.
4. Configure repository:
 - **Name** the repository.
 - **Description**. Give an optional description.
 - **Host**. Provide the name of the host machine.
 - **Repository**. Provide the name of the Vault repository.
 - **Username**. Give the user name AnthillPro will use to access the repository.
 - **Password**. Provide the password corresponding to the account specified in the Username field. Select either: Set Password or Use Script.
 - **Set Password**. If a password is associate with AnthillPro's Vault account, select this potion. Give the password in the Password field and confirm the password.
 - **Use Script (advanced)**. Give a parameterizable value that will resolve to the password. Use this instead of the Password field if the password will be stored in a secure property resolved at runtime. See [Scripting](#).
 - **Use SSL**. Check the box to use SSL when AnthillPro connects to the repository.

- **Command Path.** Provide the absolute path to the executable if not in the system path.
5. Click **Set**.
 6. Select the **Security** tab. See Manage Security.
 7. Click **Edit**, determine **permissions**, click **Save**, and then click **Done**.
 8. Click **Done**.

ViewVC

The ViewVC integrations provides CVS and Subversion users access to the ViewVC browser interface from within the AnthillPro UI. Once configured, the integration allows users to view the ViewVC navigable directory, revision, and change log listings, including specific versions of files. See ViewVC and CVS or ViewVC and Subversion.

ViewVC and CVS

The ViewVC integrations provides CVS (and CVSNT) users access to the ViewVC browser interface from within the AnthillPro UI. Once configured, the integration allows users to view the ViewVC navigable directory, revision, and change log listings, including specific versions of files.

ViewVC and CVS Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- The CVS repository must be correctly configured. See CVS.
- The ViewVC URL and repository name must be available.
- A project must be active in AnthillPro.

Set up ViewVC and CVS

Configure AnthillPro to integrate with ViewVC. Links to ViewVC will appear when this is configured.

- Historic information is not available due to the nature of the AnthillPro 3.4 integration. However, the integration will allow you to view all changes, etc., made after AnthillPro 3.5 is installed.
1. Go to **System > ViewVC** under the Integration menu.
 2. On the **ViewVC** page, click **Edit**.
 3. Give the **ViewVC Server URL**, click **Set** then **Done**.
 4. Go to **System > Repositories** from the **Project Support** menu.
 5. Select the **CVS repository** (from Item Four) from the **Repositories list**. This item requires that the CVS repository is already configured. See CVS.

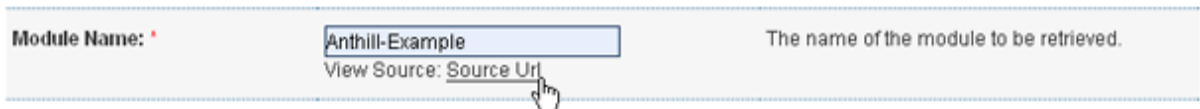
- On the **Main** tab, click **Edit** and select **ViewVC** from the drop-down menu (of the **Source Viewer** item).
 - To view the repository, follow the **Repository URL** link (of the **Source Viewer** item).
- Click **Set** then **Done**.

Using ViewVC with CVS

Once ViewVC has been set up (see Set up ViewVC and CVS), AnthillPro makes Source URL and Tags URL indexes available via ViewVC. Revision information, including detailed information on each file changed, is available on the Dashboard Changes tab.

To view Source URL and Tags URL via ViewVC:

- Go to **Administration** and select the appropriate **workflow**.
- On the workflow **Main** tab, follow the **Edit Source** link in the Source Configuration menu.
- For the Module Name item, click the **Source URL** link and ViewVC will open in a new window.

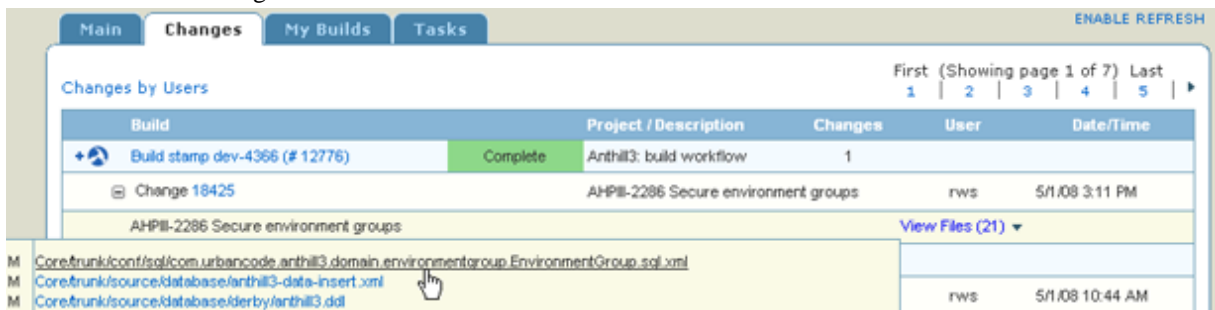


To view revision and file changes via ViewVC:

- Go to **Dashboard > Changes** tab.
- Select the **change number** link and ViewVC will open the revision in a new window.



- Expand the Change number and select the **View Files** link. Select an individual change and ViewVC will open a view of the actual change.



ViewVC and Subversion

The ViewVC integration provides Subversion users access to the ViewVC browser interface from within the AnthillPro UI. Once configured, the integration allows users to view the ViewVC navigable directory, revision, and change log listings, including specific versions of files.

ViewVC and Subversion Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See [Manage Security](#).
- The Subversion repository must be correctly configured. See [Subversion](#).
- The ViewVC URL and repository name must be available.
- A project must be active in AnthillPro.

Set up ViewVC and Subversion

Configure AnthillPro to integrate with ViewVC. Links to ViewVC will appear when this is configured. In order to use ViewVC integration, the Source Configuration **Username** field (see [Subversion](#)) must be the ViewVC repository name. It is not possible to use ViewVC if properties are set for the **Username** field.

1. Go to **System > ViewVC** under the Integration menu.
2. On the **ViewVC** page, click **Edit**.
3. Give the **ViewVC Server URL**, click **Set** then **Done**.
4. Go to **System > Repositories** from the **Project Support** menu.
5. Select the **Subversion repository** (from Item Four) from the **Repositories list**. This item requires that the Subversion repository is already configured. See [Subversion](#).
6. On the **Main** tab, click **Edit** and select **ViewVC** from the drop-down menu (of the **Source Viewer** item).
 - To view the repository, follow the **Repository URL** link (of the **Source Viewer** item).



7. Click **Set** then **Done**.

Using ViewVC with Subversion

Once ViewVC has been set up (see [Set up ViewVC and Subversion](#)), AnthillPro makes Source URL and Tags URL indexes available via ViewVC. Revision information, including detailed information on each file changed, is available on the [Dashboard Changes](#) tab.

To view Source URL and Tags URL via ViewVC:

1. Go to **Administration** and select the appropriate **workflow**.

2. On the workflow **Main** tab, follow the **Edit Source** link on the Source Configuration menu.
3. For the Source URL item, click the **Source URL** link and ViewVC will open in a new window.

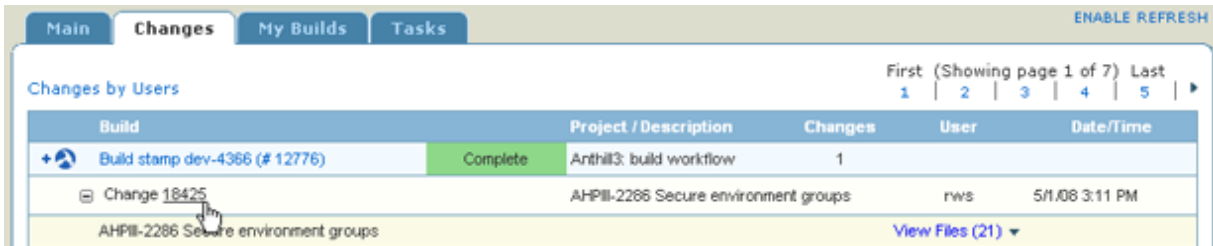


4. For the Tags URL item, click the **Tags URL** link and ViewVC will open in a new window.

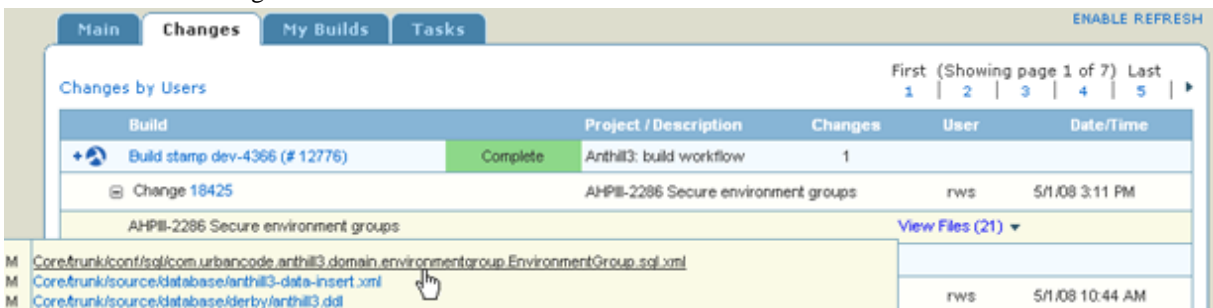


To view revision and file changes via ViewVC:

1. Go to **Dashboard** and select the **Changes** tab.
2. Select the **change number** link and ViewVC will open the revision in a new window.



3. Expand the Change number and select the **View Files** link. Select an individual change and ViewVC will open a view of the actual change.



Visual SourceSafe (VSS)

To use a Microsoft Visual SourceSafe (VSS) repository with AnthillPro, configure it by following the **Repositories** link on the System page. The configuration will allow basic information regarding VSS to be reused by several project workflows. Once configured, the repository will be listed on the **Repositories** main page.

After the main repository has been identified, your workflows can then use the VSS repository. During project creation, associate the project workflow with a specific VSS repository and create a build job that includes the VSS job steps (see **VSS Source Configuration**). If you use the **Job Wizard**, AnthillPro will automatically add these steps during job configuration; however, if you create your own job, you will need to add these steps yourself.

VSS Prerequisites

- You must have administrative permissions. See Manage Security.
- The VSS root must be available to complete the configuration. The root must include the path to the repository (where the *srcsafe.ini* file resides). This can be either a local path or a UNC path to a shared drive and *must end with a \ to signify a directory*.

Set Up VSS Repository

1. Go to **System > Repositories** from the **Project Support** menu.
2. On the **Repositories** page, click the **Create New** button.
3. Select **VSS** from the drop-down menu and click **Set**.
4. Configure repository:
 - **Name** the repository.
 - **Description**. Give an optional description.
 - **Root**. Provide the path to the repository (where the *srcsafe.ini* file resides). This could be either a local path or a UNC path to a shared drive and *must end with a \ to signify a directory*.
 - **Username**. Give the user name to use for AnthillPro to access the repository.
 - **Password**. Provide the password corresponding to the account specified in the Username field. Select either: No Password, Use Password, or Use Script.
 - **No Password**. If no password is associated with AnthillPro's VSS account, use this option.
 - **Use Password**. If a password is associate with AnthillPro's VSS account, select this potion. Give the password in the Password field and confirm the password.
 - **Use Script (advanced)**. Give a parameterizable value that will resolve to the password. Use this instead of the Password field if the password will be stored in a secure property resolved at runtime. See Scripting.
 - **Command Path**. Provide the absolute path to the *ss* executables if not in the system path.
5. Click **Set**.
6. Select the **Security** tab. See Manage Security.
7. Click **Edit**, determine **permissions**, click **Save**, and then click **Done**.
8. Click **Done**.
9. See VSS Source Configuration.

VSS Source Configuration

Once the main VSS repository is identified (see Set Up VSS Repository), configure the specific source a workflow uses. During project creation, the originating workflow is first associated with the VSS repository. Once this is done,

the Source Configuration page will automatically pop up. (However, if you choose not to configure the project source at initial creation, you can return to it by selecting the Edit Source link on the Workflow Main page.)

1. Once Git is set up with AnthillPro (see Set Up Git Repository), create a project.
2. Go to the originating workflow that was created as part of the project creation process. If you have not created an originating workflow yet, see Create a New Workflow before continuing. Configure the source on the workflow:

- **Repository.** This field is automatically populated by AnthillPro.
- **Working Directory Script.** Select the Working Directory Script from the drop-down menu. The working directory is the location where the agent is going to run its commands. For example, if the working directory is `C:\projects\Project_A` every command AnthillPro issues will be from that specific directory.

If no Working Directory Script is configured in the job, the script selected here will be used. See Working Directory Scripts.

- **Project/Branch.** Give the name of the project to use in the form of `$/project`. If you want to use a branch or module, use something like `$/project/path/to/branch`.
- **Label Script.** Give the label to use, if using labels.
- **Users to exclude from changelog.** To exclude any users from the changelog, input them here. Each excluded user must be input on a separate line. Note that user names are case sensitive.
- **File paths to exclude from changelog.** If you need to exclude file paths from the changelog, list them here, each on a separate line. Each path must be preceded by + or - to include/exclude it from the changelog.

The following wild cards may be used:

- for any subdirectory: `**/` -
- for any number of any characters except \ and /: `*` -
- for any single character / including \ and / Changelog file paths containing drive letter or any prefixes: `?` -

Make sure to include the `*` at the beginning of the path.

Example exclude paths: `-**/abc/**/*` Except for: `+**/abc/*.html`. This will exclude everything under the abc directory except for the HTML files in it.

3. Click **Save**.

Chapter 64. Build Tools

AnthillPro integrates with many build tools to support building and deploying. The integrations are implemented as job steps that allow you to execute a script written in any scripting language. For most tools, you can write your scripts within the AnthillPro UI.

When you set up your build process, the integration is added as a single step that executes your build script. For your deployment process, you use the tool to execute your deploy script that moves the artifacts to their destination.

Ant

The builder is added to your AnthillPro project as a job step. Typically, the Build step is added after a Cleanup, Populate Workspace, Stamp, and Get Dependency Artifacts step of a build job; however, your job configuration may vary (see Create a New Job). The builder is also used to perform deployments, and is added to the deployment workflow as a job step (see Create a Deployment Job).

You can write the build/deploy script within the AnthillPro UI on the Script Content tab. Or, you can write the script outside of AnthillPro and then have the server run that file.

To use Ant as part of your build or deployment process:

1. Follow the instructions for project creation given in Setting Up a Build Process if you are building a project.

Or:

Follow the instructions for Setting Up a Deployment Process if you are deploying the artifacts.

2. When you get to the Builder step, configure the Builder:

- **Name** the builder. This name will be used by the AnthillPro system.
- **Working Directory Offset.** Enter the working directory to use when executing this command. This is relative to current working directory (blank for the current). For some build systems, there will be a subdirectory named after the project or module.
- **Ant Script File.** Give the name of the Ant script file.
- **Target Name.** Enter the name of the target to run in the Ant script file.
- **Ant Properties.** Give the Ant-specific arguments, such as using `-v` for verbose output.
- **Ant Configuration.** Enter the path to the version of Ant that will be used to run the build script. If every agent that might run a build has this configured in the same location, a simple path can be entered here. More commonly, the location will vary between agents, so AnthillPro supports using a simple expression language to lookup environment variables on the agent. You may need to create an environment variable on each agent (either at those agents, or in the Agent sections under the Environment menu) with the path.
- **JVM Configuration.** Give the path to the Java version used to compile the code. If every agent that might run a build has this configured in the same location, a simple path can be entered here. More commonly, the location will vary between agents, so AnthillPro supports using a simple expression language to lookup environment variables on the agent. You may need to create an environment variable on each agent (either at those agents, or in the Agent sections under the Environment menu) with the path.
- **JVM Properties.** Enter the properties passed to the JVM.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.

- **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
3. **Ant Properties (optional).** Select the Properties tab and provide any additional properties that should be in place when the build runs. These properties are passed to the executable. It is not necessary to escape spaces or use quoted strings. Enter each property on a separate line in the following format: `name=value`.
 4. **Ant Environment Variables (optional).** Select the Environment Variables tab to provide any additional variables that should be in place when the build runs. Environment variable values may contain references to existing values in the following format: `name=${env/<NAME>};value`. If the value of the `<NAME>` variable is `value2` in the current environment, then the above example will be expanded to: `name=value2;value`. Using this technique, it is possible add an entry to `PATH` in the following manner: `PATH=my/path/entry;0`. Case is significant even on Windows systems.
 5. **Ant Script Content.** Here, you can write the build/deploy script directly in the AnthillPro UI. Click the Script Content tab to define the content of the Ant script in this step. The content will be written to file when executed.
 6. Click **Save**.

Groovy (Builder)

The builder is added to your AnthillPro project as a job step. Typically, the Build step is added after a Cleanup, Populate Workspace, Stamp, and Get Dependency Artifacts step of a build job; however, your job configuration may vary (see Create a New Job). The builder is also used to perform deployments, and is added to the deployment workflow as a job step (see Create a Deployment Job).

You can write the build/deploy script within the AnthillPro UI on the Script Content tab. Or, you can write the script outside of AnthillPro and then have the server run that file.

To use Groovy as part of your build or deployment process:

1. Follow the instructions for project creation given in Setting Up a Build Process if you are building a project.

Or:

Follow the instructions for Setting Up a Deployment Process if you are deploying the artifacts.

2. When you get to the Builder step, configure the Builder:

- **Name** the builder. This name will be used by the AnthillPro system.
- **Working Directory Offset.** Enter the working directory to use when executing this command. This is relative to current working directory (blank for the current).

- **Groovy Script File.** Give the name of the Groovy script file.
 - **Groovy Location.** Enter the path to the version of Groovy that will be used to run the build script.
 - **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
3. **Groovy Properties (optional).** Select the Properties tab and provide any additional properties that should be in place when the build runs. These properties are used as if they were passed to the Groovy executable on the command line. Enter each property on a separate line.
 4. **Groovy Environment Variables (optional).** Select the Environment Variables tab to provide any additional variables that should be in place when the build runs. Environment variable values may contain references to existing values in the following format: `name=${env/<NAME>} ; value`. If the value of the `<NAME>` variable is `value2` in the current environment, then the above example will be expanded to: `name=value2 ; value`. Using this technique, it is possible add an entry to `PATH` in the following manner: `PATH=my/path/entry;0`. Case is significant even on Windows systems.
 5. **Groovy Script Content.** Here, you can write the build/deploy script directly in the AnthillPro UI. Click the Script Content tab to define the content of the Groovy script in this step. The content will be written to file when executed.
 6. Click **Save**.

Make

The builder is added to your AnthillPro project as a job step. Typically, the Build step is added after a Cleanup, Populate Workspace, Stamp, and Get Dependency Artifacts step of a build job; however, your job configuration may vary (see Create a New Job). The builder is also used to perform deployments, and is added to the deployment workflow as a job step (see Create a Deployment Job).

To use Make as part of your build or deployment process:

1. Follow the instructions for project creation given in Setting Up a Build Process if you are building a project.

Or:

Follow the instructions for Setting Up a Deployment Process if you are deploying the artifacts.

2. When you get to the Builder step, configure the Builder:
 - **Name** the builder. This name will be used by the AnthillPro system.

- **Working Directory Offset.** Enter the working directory to use when executing this command. This is relative to current working directory (blank for the current).
 - **Make File Name.** Give the name of the Make file in the source repository. This only needs to be set if you use a non-standard Make file name.
 - **Make Targets.** Enter the target(s) to execute in the Make file. The default target will be executed if you do not specify any here. If entering multiple targets, separate each target with a space just as you would on the command line. (optional).
 - **Command-line Arguments.** Specify any additional command line arguments to pass to Make. Enter them exactly as they would appear on the command line.
 - **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
3. **Make Environment Variables (optional).** Select the Environment Variables tab to provide any additional variables that should be in place when the build runs. Environment variable values may contain references to existing values in the following format: `name=${env/<NAME>};value`. If the value of the `<NAME>` variable is `value2` in the current environment, then the above example will be expanded to: `name=value2;value`. Using this technique, it is possible add an entry to `PATH` in the following manner: `PATH=my/path/entry;0`. Case is significant even on Windows systems.
4. Click **Save**.

Maven (Builder)

The builder is added to your AnthillPro project as a job step. Typically, the Build step is added after a Cleanup, Populate Workspace, Stamp, and Get Dependency Artifacts step of a build job; however, your job configuration may vary (see Create a New Job). The builder is also used to perform deployments, and is added to the deployment workflow as a job step (see Create a Deployment Job).

If you are a Maven 2 user, you can configure AnthillPro to act as a Maven repository. This enables AnthillPro to provide extended visibility into Maven's dependency-management capabilities. See Maven 2.

To use the Maven builder as part of your build or deployment process:

1. Follow the instructions for project creation given in Setting Up a Build Process if you are building a project.

Or:

Follow the instructions for Setting Up a Deployment Process if you are deploying the artifacts.

2. When you get to the Builder step, configure the Builder:

- **Name** the builder. This name will be used by the AnthillPro system.
- **Working Directory Offset.** Enter the working directory to use when executing this command. This is relative to current working directory (blank for the current).
- **Maven File Path.** Location of the project.xml or pom.xml file relative to the project root. If your .xml file is in the root of the project, then leave this blank.
- **Maven Goal.** The name of the goal/target to run. If there are multiple goals enter them in order separated by a space
- **Maven Properties.** The name of the goal/target to run. If there are multiple goals enter them in order separated by a space
- **Maven Home.** The path to the installation of Maven that will be used.
- **Maven Version.** The version of maven that will be used. Select either 1.x or 2.x.

If you are using the Maven 2 integration to allow AnthillPro to manage dependencies, you must select Maven 2 from the drop-down menu. See Maven 2 for more about the integration.

- **Java Home.** The path to the installation of Java for Maven to use.
 - **JVM Properties.** Properties passed to the JVM.
 - **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
3. **Maven Properties (optional).** Select the Properties tab and provide any additional properties that should be in place when the build runs. These properties are used as if they were passed to the maven executable on the command line. Enter each properties on a separate line in the following format: `-Dname=value`.
4. **Maven Environment Variables (optional).** Select the Environment Variables tab to provide any additional variables that should be in place when the build runs. Environment variable values may contain references to existing values in the following format: `name=${env/<NAME>};value`. If the value of the `<NAME>` variable is `value2` in the current environment, then the above example will be expanded to: `name=value2;value`. Using this technique, it is possible add an entry to `PATH` in the following manner: `PATH=my/path/entry;0`. Case is significant even on Windows systems.
5. Click **Save**.

Maven 2

Use the Maven 2 integration to have AnthillPro act as a Maven repository for projects built in AnthillPro, as well as a caching proxy for external Maven repositories. Once the external Maven repository is configured on the System page, AnthillPro will get the artifacts from the Maven and cache them, in the form of Codestation projects, for future use. The integration allows AnthillPro to provide complete visibility into the project dependencies you manage using Maven 2.

The integration allows you to either have the artifacts retrieved from Codestation (AnthillPro's artifact repository) or directly from your Maven repositories. If pulling the artifacts directly from Maven, the system will check for AnthillPro projects first, and then check Maven. When the projects are found in Maven, AnthillPro will replicate the Maven repository in Codestation (AnthillPro's artifact management system) for future use.

When using an external Maven repository, your AnthillPro projects need to be associated with a Life-Cycle Model that contains an artifact set named "maven" (see Using a Maven-specific Life-Cycle Model), and must contain Maven-specific project properties. To use the Maven 2 integration you will need to:

1. **Modify the Maven POM file.** To use AnthillPro as a Maven repository, the POM file will need to point to the AnthillPro server.
2. **Modify build agent `settings.xml` file.** The agent must supply the user name and password used to connect to AnthillPro.
3. **Create Maven 2 Repository.** Under the Integrations menu on the **System** page, tell AnthillPro about the repository you want to create. To have AnthillPro act as more than one Maven repository, complete multiple configurations.
4. **Configure Maven 2 Projects.** The integration allows you to create both Codestation and Life-Cycle-based projects.
5. **Modify Settings on Developer Machines.** Each developer will need change their POM file to point to the correct URL and provide the correct password and username.

When a build is needed, AnthillPro will use its local cache to fulfill dependencies -- *using the most recent version*. If the appropriate dependency is not found, it will then retrieve the dependency from your other Maven repositories.

Maven 2 Prerequisites

- The build agent must be on the same machine as Maven (or Maven must be downloaded with the project).
- You must have administrative permissions to configure projects and to the System page.
- You must be able to edit the build agent's `settings.xml` file.
- You must be able to edit Maven POM file.

Modify Maven POM File

The POM file must reference what repository AnthillPro should use. To use AnthillPro as a Maven repository, the POM should be similar to the following example:

```
<project>
```

```
<distributionManagement>
  <repository>
    <id>anthill</id>
    <name>Anthill3 Repo</name>
    <url>${env.AH_WEB_URL}rest/maven2dist/${env.AH_BUILD_LIFE_ID}</url>
  </repository>
  <snapshotRepository>
    <uniqueVersion>>false</uniqueVersion>
    <id>anthill</id>
    <name>Anthill3 Repo</name>
    <url>${env.AH_WEB_URL}rest/maven2dist/${env.AH_BUILD_LIFE_ID}</url>
  </snapshotRepository>
</distributionManagement>
</project>
```

Because AnthillPro will always favor its local cache, you may need to add a setting in your POM file to *clear the cache* before the build occurs. This will ensure AnthillPro retrieves the artifacts from your other Maven repositories for the build.

Modify Build Agent Settings

The build agent `settings.xml` file must be modified to supply the user name and password (it will be eventually be obfuscated) for basic authentication with Maven.

The integration allows AnthillPro to also act as a Maven repository for developer dependency resolves, as well as collect the deployed artifacts through AnthillPro builds and for storing dependencies. Your developers will have to make a similar change in their individual setting file or overwrite the URL. See [Modify Settings on Developer Machines](#).

The `settings.xml` file should be similar to the following:

```
<settings>
  <servers>
    <server>
      <id>anthill</id>
      <username>username</username>
      <password>password</password>
    </server>
  </servers>
</settings>
```

Create Maven 2 Repository

The information given here will be used by your AnthillPro projects. If you have more than one Maven repository, create a new repository for each.

1. Go to **System > Maven** under the Integrations menu.

On the Maven Repositories page, determine the dependency trigger type and create new repositories.

2. **Dependency Trigger Type.** When AnthillPro detects dependencies between AnthillPro projects through Maven, the type of build trigger that will be automatically created is configured globally, and is based on your current practices defined in your POM file. To set a dependency-trigger strategy, click **Edit** and choose either No Trigger, Push or Pull.

- **No Trigger.** AnthillPro will not use a dependency trigger.
- **Push.** See Pushing Builds for more on AnthillPro pushing dependency triggers.
- **Pull.** See Pulling Builds for more on AnthillPro pulling dependency triggers.

3. **Maven Repositories.** Click **Create Repository** and configure the repository:

- **Repository Name.** Give the name of the repository AnthillPro should retrieve the artifacts from. This is the name set in Maven.
- **Description (optional).** Provide a unique description of this repository.
- **URL.** Give the exact URL of the Maven repository AnthillPro will retrieve the artifacts from.
- **Proxy.** If you need to use a proxy, select it from the drop-down menu.

If AnthillPro must connect to a Maven 2 repository via a proxy (e.g., it is public or behind a firewall), set up a proxy server before continuing. Once the AnthillPro proxy is set up, associate all the Maven 2 repositories with the AnthillPro proxy that will connect to them. If you have multiple Maven repositories that must be connected via a proxy, configure a separate proxy for each. To configure a proxy:

- a. Select the Maven link under the Integration menu on the System page. Follow the Proxies link on the Maven Settings page.
- b. See Create Server Proxy.

4. Click **Save**.

5. To add another repository, repeat *Items 3 and 4* then click **Done**.

6. See Configure Maven 2 Projects.

Configure Maven 2 Projects

Basic project configuration is the same as outlined in the Setting Up a Build Process and Using Codestation Projects sections. The instructions below assume you are using the Maven Builder integration for builds and the "Maven Projects" Life-Cycle Model.

Setting project properties. The Maven 2 integration requires you to set specific AnthillPro project properties and use a specialized Life-Cycle Model.

- **Life-Cycle-based projects (for builds).** Create a `maven.groupId` property containing the project's Maven `groupId` **and** a `maven.artifactId` property containing the project's Maven `artifactId`.

When Maven publishes artifacts to AnthillPro and parses the published POM file for dependencies, the system will automatically set the correct `maven.version` property on the Build Life.

- **Codestation projects.** Create a `maven.groupId` property containing the project's Maven `groupId` **and** a `maven.artifactId` property containing the project's Maven `artifactId`.

Make sure that the Codestation project includes a POM file if you manually create one. Otherwise, the integration may fail during the resolve or build.

Using a Maven-specific Life-Cycle Model. Any project using the Maven 2 integration must include an artifact set called "maven". The artifacts will then be stored in a folder named 'Maven Projects' when using an external Maven repository. You can either use the "Maven Projects" Life-Cycle Model that ships with AnthillPro (go to **System > Project Support > Life-Cycle Models > Maven Projects** for an example), or add the "maven" artifact set to an existing Life-Cycle Model.

Modify Settings on Developer Machines

The integration allows AnthillPro to act as a Maven repository for developer dependency resolves, as well as collect the deployed artifacts through AnthillPro builds and for storing dependencies. Each developer will need to change their POM file to point to their local repository URL. It should look something like this:

```
<project>
  <repositories>
    <repository>
      <id>anthill</id>
      <name>Anthill3 Repo</name>
      <url>http://anthillpro.yourcompany.com:8080/rest/maven2</url>
    </repository>
  </repositories>
</project>
```

In addition, each developer will need to modify their `settings.xml` file (similar to what was outlined in the Modify Build Agent Settings section) to provide the correct username and password. It should look something like this:

```
<settings>
  <servers>
    <server>
      <id>anthill</id>
      <username>username</username>
      <password>password</password>
    </server>
  </servers>
</settings>
```

MSBuild

The builder is added to your AnthillPro project as a job step. Typically, the Build step is added after a Cleanup, Populate Workspace, Stamp, and Get Dependency Artifacts step of a build job; however, your job configuration may vary (see Create a New Job). The builder is also used to perform deployments, and is added to the deployment workflow as a job step (see Create a Deployment Job).

You can write the build/deploy script within the AnthillPro UI on the Script Content tab. Or, you can write the script outside of AnthillPro and then have the server run that file.

To use MSBuild as part of your build or deployment process:

1. Follow the instructions for project creation given in Setting Up a Build Process if you are building a project.

Or:

Follow the instructions for Setting Up a Deployment Process if you are deploying the artifacts.

2. When you get to the Builder step, configure the Builder:

- **Name** the builder. This name will be used by the AnthillPro system.
 - **Working Directory Offset.** Enter the working directory to use when executing this command. This is relative to current working directory (blank for the current).
 - **Command Path.** The full path to msbuild.exe including msbuild.exe (blank if it is on the path).
 - **Build File.** The name of the build file to execute.
 - **Targets.** The name of the targets to run. Multiple targets can be separated with semi-colons or commas.
 - **Verbosity Level.** The verbosity level.
 - **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
3. **MSBuild Properties (optional).** Select the Properties tab and provide any additional properties that should be in place when the build runs.
- **Properties.** These properties are passed to the MSBuild executable. It is not necessary to escape spaces or use quoted strings. Enter each property on a separate line in the following format: `name=value` (e.g., `WarningLevel=2` will become `/p:WarningLevel=2` on the command line).
 - **Parameters.** These parameters are directly passed to the MSBuild executable. It is not necessary to escape spaces or use quoted strings. Enter each parameter on a separate line (e.g., `/clp:PerformanceSummary`).
4. **MSBuild Environment Variables (optional).** Select the Environment Variables tab to provide any additional variables that should be in place when the build runs. Environment variable values may contain references to existing values in the following format: `name=${env/<NAME>} ; value`. If the value of the `<NAME>` variable is `value2` in the current environment, then the above example will be expanded to: `name=value2 ; value`. Using this technique, it is possible add an entry to `PATH` in the following manner: `PATH=my/path/entry;0`. Case is significant even on Windows systems.
5. **MSBuild Script Content.** Here, you can write the build/deploy script directly in the AnthillPro UI. Click the Script Content tab to define the content of the Groovy script in this step. The content will be written to file when executed.
6. Click **Save**.

Nant

The builder is added to your AnthillPro project as a job step. Typically, the Build step is added after a Cleanup, Populate Workspace, Stamp, and Get Dependency Artifacts step of a build job; however, your job configuration may

vary (see Create a New Job). The builder is also used to perform deployments, and is added to the deployment workflow as a job step (see Create a Deployment Job).

You can write the build/deploy script within the AnthillPro UI on the Script Content tab. Or, you can write the script outside of AnthillPro and then have the server run that file.

To use Nant as part of your build or deployment process:

1. Follow the instructions for project creation given in Setting Up a Build Process if you are building a project.

Or:

Follow the instructions for Setting Up a Deployment Process if you are deploying the artifacts.

2. When you get to the Builder step, configure the Builder:

- **Name** the builder. This name will be used by the AnthillPro system.
- **Working Directory Offset.** Enter the working directory to use when executing this command. This is relative to current working directory (blank for the current).
- **Nant Script File.** The name of the Nant script file.
- **Target Name.** The name of the target(s) to run in the Nant script file.
- **Nant Properties.** Nant-specific arguments, such as `-ext`.
- **Nant Location.** The path to the version of Nant that will be used to run the build script.
- **Mono Location.** The path to the version of mono to run Nant to run with. If this is empty or evaluates to nothing on the agent, mono will not be used.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

3. **Nant Properties (optional).** Select the Properties tab and provide any additional properties that should be in place when the build runs. These properties are used as if they were passed to the Nant executable on the command line. Enter each properties on a separate line in the following format: `-D:name=value`.

4. **Nant Environment Variables (optional).** Select the Environment Variables tab to provide any additional variables that should be in place when the build runs. Environment variable values may contain references to existing values in the following format: `name=${env/<NAME>};value`. If the value of the `<NAME>` variable is `value2` in the current environment, then the above example will be expanded to: `name=value2;value`. Using this technique, it is possible add an entry to `PATH` in the following manner: `PATH=my/path/entry;0`. Case is significant even on Windows systems.

5. **Nant Script Content.** Here, you can write the build/deploy script directly in the AnthillPro UI. Click the Script Content tab to define the content of the Groovy script in this step. The content will be written to file when executed.
6. Click **Save**.

Shell Builder

The builder is added to your AnthillPro project as a job step. Typically, the Build step is added after a Cleanup, Populate Workspace, Stamp, and Get Dependency Artifacts step of a build job; however, your job configuration may vary (see Create a New Job). The builder is also used to perform deployments, and is added to the deployment workflow as a job step (see Create a Deployment Job).

To use the Shell builder as part of your build or deployment process:

1. Follow the instructions for project creation given in Setting Up a Build Process if you are building a project.

Or:

Follow the instructions for Setting Up a Deployment Process if you are deploying the artifacts.

2. When you get to the Builder step, configure the Builder:

- **Name** the builder. This name will be used by the AnthillPro system.
- **Working Directory Offset.** Enter the working directory to use when executing this command. This is relative to current working directory (blank for the current).
- **Interpreter (optional).** By default, the following platform-specific command-line behavior will be used when this field is left blank:
 - *Windows:* .bat file interpreter.
 - *VMS:* .com file interpreter.
 - *Other systems:* the shell specified by 'anthill3/shell' agent variable if present (see Agent Properties). *nix systems may override this by having the Command Line starts with a #! sequence.

If you need AnthillPro to use a different interpreter to evaluate the command, specify it here.

- **Shell Script.** Enter the script to be executed. The script is going to be passed to and run on the native shell of the selected agent.
- **Daemon.** Check this option to run the command in the background while permitting the step to complete immediately. AnthillPro does not capture the output of a daemon command.
 - **Output File.** Enter the path of the file to which the command output is directed. Leave blank to discard the output. Setting a value for this option does not cause AnthillPro to capture the output as part of the step.
- **Impersonation.** Check the box to run the builder as an impersonated user. *This option requires the agent run as root on Unix or Local System on Windows.* Note that any commands available to the superuser will be available to the agent when impersonation is used.

Impersonation is only necessary when a user may have permission to do one part of a process but does not have permission to execute another part of the process. For example, take the common application-specific IDs "oracle" and "websphere." In order to run a database update script, the user needs to be the "oracle" user;

however, to update the application the user needs to be "websphere." Using impersonation, the same user can run the update script as the "oracle" user and also update the application as the impersonated "websphere" user. This will allow you to combine these steps into a single process.

- **User.** Give the user name for the impersonated user.
 - **Password.** Give the password for the impersonated user.
 - **Confirm.** Give the password again.
 - **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
3. **Shell Builder Environment Variables (optional).** Select the Environment Variables tab to provide any additional variables that should be in place when the build runs. Environment variable values may contain references to existing values in the following format: `name=${env/<NAME>};value`. If the value of the `<NAME>` variable is `value2` in the current environment, then the above example will be expanded to: `name=value2;value`. Using this technique, it is possible add an entry to `PATH` in the following manner: `PATH=my/path/entry;0`. Case is significant even on Windows systems.
4. Click **Save**.

Visual Studio (headless mode)

The builder is added to your AnthillPro project as a job step. Typically, the Build step is added after a Cleanup, Populate Workspace, Stamp, and Get Dependency Artifacts step of a build job; however, your job configuration may vary (see Create a New Job). The builder is also used to perform deployments, and is added to the deployment workflow as a job step (see Create a Deployment Job).

To use the Visual Studio builder as part of your build or deployment process:

1. Follow the instructions for project creation given in Setting Up a Build Process if you are building a project.

Or:

Follow the instructions for Setting Up a Deployment Process if you are deploying the artifacts.

2. When you get to the Builder step, configure the Builder:

- **Name** the builder. This name will be used by the AnthillPro system.

- **Working Directory Offset.** Give the working directory to use when executing this command. This is relative to current working directory (blank for the current).
 - **Solution File.** The location of the solution file.
 - **Command Path.** The path to the devenv executable file in the Visual Studio Installation.
 - **Build Command.** The Visual Studio Command to Run.
 - **Mode.** Whether you are building in Debug or Release mode.
 - **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
3. **Visual Studio Properties (optional).** Select the Properties tab and provide any additional properties that should be in place when the build runs. These properties are used as if they were passed to the devenv executable on the command line. Enter each property on a separate line.
 4. **Visual Studio Environment Variables (optional).** Select the Environment Variables tab to provide any additional variables that should be in place when the build runs. Environment variable values may contain references to existing values in the following format: name=\${env/<NAME>};value. If the value of the <NAME> variable is value2 in the current environment, then the above example will be expanded to: name=value2;value. Using this technique, it is possible add an entry to PATH in the following manner: PATH=my/path/entry;0. Case is significant even on Windows systems.
5. Click **Save**.

Chapter 65. Testing Tools

The testing tool integrations allow you to run a suite of automated tests either on your build workflow or as a secondary process -- in addition to gathering data from tests that are invoked with your build script. Most testing integrations capture the data generated by the tests, and make that information available through the UI. This allows you to perform trending over time, to see which tests are failing, track new tests, etc.

The integrations, implemented as job steps, generally enable AnthillPro to:

- **Run tests.** This step allows AnthillPro to run a test or suite of tests. If your tests are invoked from your build scripts (e.g., using JUnit), the integration will allow AnthillPro to gather the data and store it in the AnthillPro data warehouse.
- **Capture data and publish a report.** This step collects the test results and other information and then stores it in the AnthillPro data warehouse. This allows you to generate reports and make them available through the AnthillPro UI.
- **Tool-specific commands.** For most testing tools, AnthillPro can also perform tasks only supported by that particular tool.

Agitar

Run Agitate and generate the Dashboard Report with the Agitar integration. AnthillPro users can also add information about JUnit test results to a data warehouse.

In order to use the integration, the Agitar `.arx` project-file name and the Eclipse directory path must be available. If using the JUnit option and running JUnit tests separate from the Agitation process, the directory with JUnit results (XML format) must also be available.

The integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to a workflow (under the Definition tab) as either a standalone workflow or as part of a build job.

Agitar steps:

- **Agitate.** Runs the agitate process against code. See Configure Agitate Step.
- **Publish Dashboard Report.** Publishes the results of the Agitar Dashboard report under the Reports tab. Optionally, this step can be used to automatically generate the report. See Configure Agitar Publish Dashboard Report Step.

This tutorial will follow a simple project configuration that uses the Agitar Agitate and Publish Dashboard Report steps as part of a build workflow. The example in this tutorial uses Subversion, but the basic configuration is similar for any repository type. Your jobs will vary, but the Agitar integration is added as a job step similar to what is described below.

Though the example goes through the manual creation of a build job, it is possible to use the Job Wizard to create a build job and then manually configure a second job to run as part of the same workflow. Optionally, the Agitar integration may also be configured to run as a standalone workflow, typically requiring only a Set Working Directory step; however, depending on the repository used, additional steps may be required.

Agitar Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- A project with at least one Build Life must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See Using Life-Cycle Models.
- The Agitar `.arx` project file name and the Eclipse directory path must be available. If using the JUnit option and running JUnit tests separate from the Agitation process, the directory with JUnit results (XML format) must also be available.

Configure Agitate Step

The Agitate step is included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the typical Build job. Though the example below runs the Agitate as part of a build workflow, it is possible to use the Agitar integration as part of a standalone workflow. If using this option, typically only a Set Working Directory step is required; however, depending on the repository used and your particular development environment, additional steps may also be required.

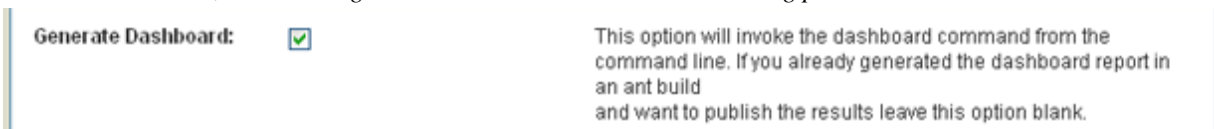
1. Go to **Administration** page, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. **Agitate**. Select the **Insert After** icon of the step prior to the point where the Agitar step is to be included (Artifact Delivery step in this example). Go to **Test > Agitar**, select the **Agitate** step, and click **Select**.
 - **Name** the step (required).
 - **Description**. Provide a short description.
 - **Project File Name (required)**. Give the Agitar `.arx` project-file name. Must be the same for both the Agitate and Publish Report steps.
 - **Eclipse Home (required)**. Provide the path where Eclipse is installed. Must be the same for both the Agitate and Publish Report steps.
 - **Command Path**. Give the path to the Agitar Command Line Interface. Must be the same for both the Agitate and Publish Report steps.
 - **Base Checkpoint**. Provide the checkpoint to base the Agitation off of.
 - **Class Regex**. Give one or more regular expressions to defining the classes to be processed. If more than one expression is used, *separate them with spaces*.
 - **Class List**. Provide the path to the file containing a list of classes to process. *Each class name must be on a separate line* (a regular expression may be used).
 - **Agitate Timeout**. Give the Agitate timeout limit in Seconds.
 - **Log Level**. Select the Agitate logging level (Fine, Info, Status, Warning, or Severe) from the drop-down menu. If none is selected, "Info" will be used by default.

- **Solving Level.** Select the Agitate solving level (Normal, Extended, or Aggressive) from the drop-down menu.
- **Has Config.** Check the box to only include classes with defined assertions, factory assignments, or object properties.
- **Merge Coverage Info.** Check the box to merge the coverage information for this run with existing coverage values for the project.
- **Reach Time Limit.** Check the box to continue exercising the methods in each class until the time limit for that class has been reached.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
- Click **Save**.

Configure Agitar Publish Dashboard Report Step

Include the Publish Dashboard Report step after the Agitate step (see Configure Agitate Step). Though the example below runs the Agitar steps as part of a build workflow, it is possible to use the Agitar integration as part of a standalone workflow. If using this option, typically only a Set Working Directory step is required; however, depending on the repository used and your particular development environment, additional steps may also be required.

- Checking the Generate Dashboard option allows AnthillPro to invoke the Agitar Dashboard command from the command line. Once selected, additional options appear on the screen. *If the Dashboard Report is already generated in an Ant build, not checking the box will result in those results being published.*



1. Select the **Insert After** icon of the Agitate step. Go to **Test > Agitar**, select the **Publish Dashboard Report** step, and click **Select**.
 - **Name** the step.
 - **Description.** Provide a short description.
 - **Report Name.** Give the name for this report. If no name is given, the Step Name will be used as default.

- **Reports Output Directory.** Enter the location of the Agitar Dashboard management reports.
- **Test Results Directory.** Provide the directory with JUnit results (they must be in XML format). This option is only required if you JUnit tests are run separately from Agitation.
- **Generate Dashboard.** Check the box to invoke the Agitar Dashboard command from the command line. If using this option, proceed to Item Two.

Do not check the box if the Agitar Dashboard report has already been generated during the Ant build and those results are to be published. If not using the Generate Dashboard option, proceed to Item Seven.

- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options. See Show Additional Options.
 - If the Generate Dashboard option was not checked, Click **Save**.
2. If the **Generate Dashboard** option is used, provide the following information (all the fields below should be familiar to regular Agitar users):

- **Project.** Give the name of the Agitation `.arx` project file (e.g., `MyProject.arx`). Must be the same for both the Agitate and Publish Report steps.
- **Eclipse Home (required).** Provide the path where Eclipse is installed. Must be the same for both the Agitate and Publish Report steps.
- **Command Path.** Give the path to the Agitar Command Line Interface. Must be the same for both the Agitate and Publish Report steps.
- **Project Name.** Enter the project name to place at the top of generated reports.
- **Checkpoint.** Provide the checkpoint file (with the extension `.ddf`) to compare the most recent test results with. If the value does not point to a specific checkpoint, the closest checkpoint to the date specified is used.
- **Log Level.** Select the Agitate logging level (Fine, Info, Status, Warning, or Severe) from the drop-down menu. If none is selected, "Info" will be used by default.
- **Server Root URL.** Enter the root URL where generated reports can be accessed.
- **Test Assignments File.** Provide the XML file that maps test classes to project classes.
- **Trend Window.** Determine the period of time the Management Dashboard reports will cover. Specify a checkpoint for the starting point of the window (current time is the ending point).

3. **Show Class Path Options.** Select the Show Class Path Options link to configure more options.

- **Target Classpath.** Give the search path for target (project) classes.
- **Test Classpath.** Provide the search path for test classes.
- **Exclude Classes.** Enter a regular expression specifying the classes to exclude from the Dashboard.
- **Class List.** Give the file that contains a list of classes to be processed, *with each class name on a separate line*. Regular expressions may be used to specify classes.
- **Class Regex.** Provide a list of one or more regular expressions, *separated by spaces*, defining the classes to process.

4. **Show Directory Options.** Select the Show Directory Options link to configure more options.

- **Agitar Path (for virtual Dashboards only).** Give the search path for Agitar directories of projects for a virtual project Dashboard, *separated with semicolons*.
 - **Config Path.** Provide the checkpoint file (with the extension `.ddf`) to compare the most recent test results with. If the value does not point to a specific checkpoint, the closest checkpoint to the date specified will be used.
 - **Coverage Results Directory (required if running JUnit separate from Agitation).** Enter the directory with coverage results in `.acov` or `.aout` files, generated either by AgitarOne or by the instrument Ant task.
 - **Data Directory.** Give the directory to save checkpoint files (extension `.ddf`) in.
 - **Lib Path.** Provide the search path for required library files of the project.
 - **Source Path (JUnit-only projects).** Provide the location of source code for generation of class coverage details. This option is ignored if an agitation project (`.arx`) file is specified.
5. **Show Target Options.** Select the Show Target Options link to configure more options.
- **Method Risk Threshold.** Determine the acceptable threshold value for method level.
 - **Target High Risk Classes.** Set the target percentage classes with risky methods for the project.
 - **Target Classes With Testpoints.** Enter the percentage of project classes that must have at least one test point.
 - **Target Coverage.** Give the target coverage percentage for this project.
 - **Target Methods With Test Points.** Set the percentage of methods that must have at least one test point.
 - **Target Test Points.** Give the target number of test points for this project.
6. **Show Args Options.** Select the Show Args Options link to configure more options.
- **Has Config.** Check the box to only includes classes that have any assertions, factory assignments, or object properties defined.
 - **Override Has Config.** Check the box to includes all classes (in an Agitation project) in the generated Management Dashboard reports, whether the classes contain configuration information or not. *This option takes precedence over the **Has Config** option.* See above.
 - **Coverage Details.** Check the box to includes all classes (in an Agitation project) in the generated Management Dashboard reports, whether the classes contain configuration information or not. *This option takes precedence over the **Has Config** option.* See above.
 - **No Coverage Details.** Check the box to overrides the inclusion of detailed coverage statistics in generated Management Dashboard reports. *This option takes precedence over the **Coverage Details** option.* See above.
 - **Generate XML Dashboard.** Check the box to create an XML file in the directory specified for the **Reports Output Directory** option (containing the Dashboard results). See Reports Output Directory.
 - **Rollup.** Check the box to specify the Dashboard is for a virtual project. *If Rollup is specified, **Agitar Path** is required.* See Show Directory Options.
 - **Use Dashboard Config.** Check the box to use the Dashboard configuration.
7. **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options. See Show Additional Options.
8. Click **Save**.

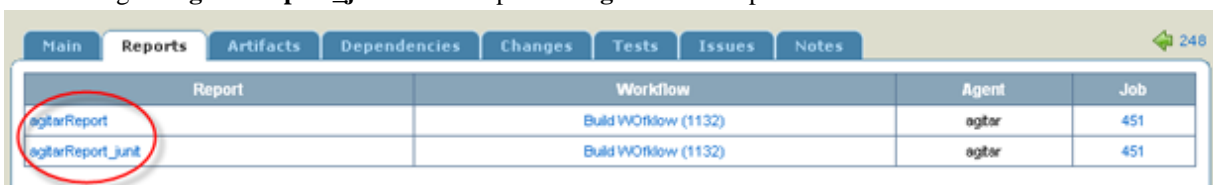
Add Agitar Job to Workflow

The Job created (see Configure Agitate Step) must be executed as part of a workflow. This section will assume that an originating workflow has already been configured, and will cover the process of adding the Agitar Build job to the appropriate workflow. Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the Agitar integration.

1. Go to **Administration**, select the **project**, and select the **build** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Build** job created in the Configure Agitate Step section, a **job pre-condition** script, and click **Insert Job**.

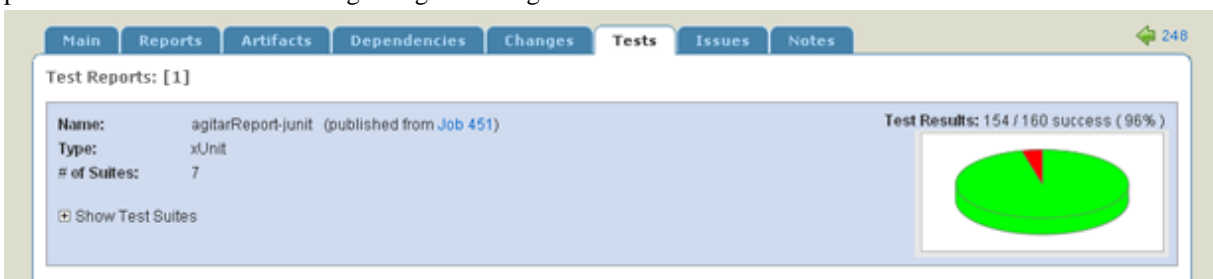
Run Agitar Workflow and View Reports

1. Go to the **Dashboard** and select the **workflow** created in the Add Agitar Job to Workflow section.
2. On the workflow **Main** page, click the **Build** button for the workflow.
3. Once the **workflow** has completed, select the appropriate **Build Life** and click the **Reports** tab.
4. Select a **link** to view either the **agitateReport** or the **agitateReport_junit**.
 - Selecting the **agitateReport** link to open the **Agitar Dashboard**.
 - Selecting the **agitateReport_junit** link to open the **Agitar/Junit** report.



Report	Workflow	Agent	Job
agitarReport	Build Workflow (1132)	agitar	451
agitarReport_junit	Build Workflow (1132)	agitar	451

5. Click the **Tests** tab for metrics on all the **agitateReport_junit** reports. Selecting the Show Test Suites link provides detailed information regarding all the Agitar/Junit tests run on the Build Life.



6. To drill down on each Agitar step on the Build Life Summary page, see Trace a Build Life to Source.

CppUnit

In order to integrate with CppUnit, your build scripts must run CppUnit and have it generate the test output XML files in the format *.xml. AnthillPro takes the information from the output file and makes it available through the UI and (optionally) via e-mail.

Add CppUnit Report Publisher to Build Job

Identify the CppUnit output files by adding a **CppUnit Report** step to the build job. This allows AnthillPro to store the data and make it available to users.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.

Before proceeding to Item Four, ensure CppUnit is associated with the build script. See CppUnit documentation [<http://cppunit.sourceforge.net/doc/1.8.0/index.html>].

4. **CppUnit Report**. Select the **Insert After** icon of the step prior to the point where the CppUnit step is to be included. Go to **Test > CppUnit**, select the **CppUnit Report** step, and click **Select**.

- **Name** the step.
- **Description**. Provide a short description.
- **Report Name**. Give the name for this report (default is same as step name) to appear on the Dashboard.
- **Source Directory**. Provide the directory where the CppUnit data files will be retrieved from.
- **Include Patterns**. Give the file name patterns that describe the files to be retrieved. Each include pattern must be entered on a separate line.

You can also use the following wild cards to tell AnthillPro what to include:

- ****** Indicates include every directory within the base directory.
- ***** Used to include every file. So, if you use *.zip, the files matching this pattern will be included.
- ****/*** Tells AnthillPro to retrieve the entire file tree underneath the base directory.
- **Exclude Patterns**. Provide the file name patterns identifying the files that will NOT be retrieved. This field is set in the same way as the Include Patterns field, only you are telling AnthillPro what NOT to include.
- **Show Additional Options (advanced)**. Select the Show Additional Options link to configure more options.
 - **Is Active**. Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script**. From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures**. Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript**. Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.

- **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

5. Click **Save**.

View CppUnit Reports

Once a build has completed, go to the new **Build Life** and select the **Reports** and **Tests** tabs for information on the tests run.

E-mailing CppUnit Results

Since AnthillPro is aware of CppUnit, the results can be included in build notifications, just like any other piece of information. For example, you may want to automatically take action depending on CppUnit results. Some common examples are to e-mail the team if a failure occurs; e-mail the QA team leader if there are no errors; or e-mail someone if the tests-per-second metric goes below a target amount.

To edit a notification template, go to the System page and choose the **Notification Templates** link from the Notification menu. See Managing Notifications.

1. Lookup the **CppUnit report(s)** and put them in the **Velocity** context.
2. Once the **CppUnitReports** are available to the **Velocity** template, you can do whatever you like with them.
3. Included in an HTML-friendly e-mail template, this will produce a nice little report.

JUnit

In order to integrate with JUnit (or TestNG), your build scripts must run JUnit and have it generate the test output XML files in the format of `TEST-*.xml`.

In Ant, the example project is configured to generate these files during the Ant JUnit task:

```
<junit printsummary="on" haltonfailure="false" fork="true">
  <classpath refid="tests.classpath"/>
  <formatter type="xml"/>
  <batchtest todir="">
    <fileset dir="" includes="**/*Test*.class"/>
  </batchtest>
</junit>
```

- The Ant JUnit report task will delete the report files as part of its transformation. If you use that task, make a copy of the `TEST-*.xml` file for AnthillPro to use.

AnthillPro reads the output XML files to understand what happened during the JUnit tests. To tell AnthillPro about those files, you need to: (a.) identify where the files are generated in the project; and (b.) add a **JUnit Report** step on the project build-job configuration.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.

2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.

Before proceeding to Item Four, ensure JUnit is associated with the build script. See JUnit documentation [<http://junit.sourceforge.net/>].

4. **JUnit Report**. Select the **Insert After** icon of the step prior to the point where the JUnit step is to be included. Go to **Test > JUnit**, select the **JUnit Report** step, and click **Select**.

- **Name** the step.
- **Description**. Provide a short description.
- **Report Name**. Give the name for this report (default is same as step name).
- **Source Directory**. Give the directory where the JUnit data files will be retrieved from. This is relative to the working directory. For example, give `..\..\directoryname\`.
- **Include Patterns**. Give the file name patterns that describe the files to be retrieved. Each include pattern must be entered on a separate line.

You can also use the following wild cards to tell AnthillPro what to include:

- `**` Indicates include every directory within the base directory.
- `*` Used to include every file. So, if you use `*.zip`, the files matching this pattern will be included.
- `**/*` Tells AnthillPro to retrieve the entire file tree underneath the base directory.
- **Exclude Patterns**. Provide the file name patterns identifying the files that will NOT be retrieved. This field is set in the same way as the Include Patterns field, only you are telling AnthillPro what NOT to include.
- **Show Additional Options (advanced)**. Select the Show Additional Options link to configure more options.
 - **Is Active**. Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script**. From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures**. Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript**. Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout**. Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

5. Click **Save**.

Once the new settings are saved, running the build reveals there is a new report stored in the **Tests and Reports** tabs of the Build Life (Build Life 127 in the example below). This is a basic report checking the JUnit results. More importantly, the JUnit results are now exposed in a meaningful way.

Test Reports: [1]

Name: (published from Job 575) Test Results: 1 / 2 success (50%)

Type: xUnit

of Suites: 1

Hide Test Suites

WidgetTestCase 1 / 2 success (50%)

example.WidgetTestCase	testWidget	success	0.000 s
example.WidgetTestCase	testFailure	failure	0.000 s

junit.framework.AssertionFailedError: No reason, just wanted to fail
at example.WidgetTestCase.testFailure(WidgetTestCase.java:56)

E-mailing JUnit Results

Since AnthillPro is now aware of the JUnit results, they can be included in build notifications, just like any other piece of information. To edit a notification template, go to the System page and choose the **Notification Templates** link from the Notification menu. See Managing Notifications.

1. Lookup the **JUnit report(s)** and put them in the **Velocity** context by adding the following to the context script:

```
WorkflowCase workflow = (WorkflowCase)context.get("workflow");
JUnitReportArray junitReportArray = JUnitReportHelper.getJUnitReportArray(workflow);
context.put("junitReportArray", junitReportArray);
```

2. Once the **JUnitReports** are available to the **Velocity** template, you can do whatever you like with them. As an example, enter a template that prints out some summary information, as well as a list of every JUnit test that did not pass.

```
#foreach($junitReport in $junitReportArray)
<h1>JUnit Test Results</h1>
<H2>Test Summary:</H2>
<div class="data-table-container">
<table class="data-table" cellpadding="4" cellspacing="1" width="100%">
  <tr class="data-table-head">
    <th>Total:</th>
    <th>Pass:</th>
    <th>Fail:</th>
    <th>Errors:</th>
    <th>Success Rate:</th>
    <th>Time:</th>
  </tr>
  <tr bgcolor="#ffffff">
    <td align="center" style="font-weight: bold;font-size: 14px;">
      $junitReport.testSummary.tests
    </td>
  </tr>
</div>
```

```

<td align="center" style="color: green;font-weight: bold;font-size:
  14px;">
  $junitReport.testSummary.passingTests
</td>
<td align="center" style="color: red;font-weight: bold;font-size:
  14px;">
  $junitReport.testSummary.failures
</td>
<td align="center" style="color: red;font-weight: bold;font-size:
  14px;">
  $junitReport.testSummary.errors</B>
</td>
<td align="center" style="font-weight: bold;font-size: 14px;">
  $fn.formatNumber($junitReport.testSummary.successPercentage ,
    '##.##')
</td>
<td align="center" style="font-weight: bold;font-size: 14px;">
  $fn.formatNumber($fn.duration($junitReport.testSummary.time),
    '##0.0') secs
</td>
</tr>
</table>
</div>

#foreach( $suite in $junitReport.suiteResultArray)
  #if( $suite.passingTestCount < $suite.tests )
<div class="data-table-container">
  <h3> Failed tests in $suite.name </h3>
<table class="data-table" cellpadding="4" cellspacing="1" width="100%">
  #foreach( $case in $suite.testCaseResults )
    #if( $case.errorCount > 0 )
      #foreach( $error in $case.errorArray )
        <tr bgcolor="#ffffff">
          <td valign="top" style="color: red;font-weight: bold;" width="275">
            $case.name<br>
            (error)</td>
          <td valign="top" style="color: red;"><B>$error.type</b><br>
            $error.content
          </td>
        </tr>
      #end
    #end
  #if( $case.failureCount > 0 )
    #foreach( $error in $case.failureArray )
      <tr bgcolor="#ffffff">
        <td valign="top" style="color: red;font-weight: bold;" width="275">
          $case.name<br>
          (failure)</td>
        <td valign="top" style="color: red;"><B>$error.type</b><br>
          <b>$error.message</b><br/> $error.content
        </td>
      </tr>
    #end
  #end
</table>
</div>
#end
#end
#end

```


3. Included in an HTML-friendly e-mail template, this will produce a nice little report. The example project has one failing test and one passing test. That produces the following in an e-mail.

JUnit Report

Test Summary:

Total:	Pass:	Fail:	Errors:	Success Rate:	Time:
2	1	1	0	50%	0.2 secs

Class Summary:

Package:	Name:	Tests:	Errors:	Failures:	Time:
example	WidgetTestCase	2	0	1	0.2 secs

[Back to Top](#)

Test Detail for: example.WidgetTestCase

Name	Status	Details	Time
testWidget	Success		0.0 secs
testFailure	Failure	No reason, just wanted to fail junit.framework.AssertionFailedError: No reason, just wanted to fail at example.WidgetTestCase.testFailure(WidgetTestCase.java:56)	0.0 secs

Making Decisions Based On JUnit Results

You may want to automatically take action depending on JUnit results. Some common examples are to e-mail the team if a failure occurs; e-mail the QA team leader if there are no errors; or e-mail someone if the tests-per-second metric goes below a target amount.

- **Below is a notification event-selector script to notify when JUnit tests fail:**

```
import com.urbancode.anthill3.domain.workflow.*;

result = false;
if (event instanceof WorkflowEvent &&
    event.getCase().isComplete() &&
    JUnitReportHelper.getJUnitReportArray(event.getCase())
        !=null) {

    workflow = event.getCase();
    junitReports =
        JUnitReportHelper.getJUnitReportArray(workflow);
    for (int i = 0; i < junitReports.length; i++) {
        summary = junitReports[i].getTestSummary();
        if(summary.getPassingTests() != summary.getTests()) {
            result = true;
        }
    }
}
return result;
```

- **Script that e-mails when some tests take too long:**

```
import com.urbancode.anthill3.domain.workflow.*;

double testPerSecondNeeded = 100;
result = false;
if (event instanceof WorkflowEvent &&
    event.getCase().isComplete() &&
    JUnitReportHelper.getJUnitReportArray(event.getCase())
        !=null) {

    workflow = event.getCase();
    junitReports =
        JUnitReportHelper.getJUnitReportArray(workflow);
    for (int i = 0; i < junitReports.length; i++) {
        summary = junitReports[i].getTestSummary();
        if(summary.getTestsPerSecond() < testPerSecondNeeded){
            result = true;
        }
    }
}
return result;
```

Next Steps (JUnit)

The JUnit report generated by some build scripts can be valuable as well. Since its creation tends to delete the required XML files, copy those files to another location in the script and generate that report. Then add an AnthillPro publisher to publish the resulting JUnit HTML report for review by the team.

- The number of tests that should run per second may differ by project. A different case selector could be made for every project, but it would make more sense to add to each project a project property with the number of tests that should run each second, and reference that property in the case selection script.
- Another option would be to create a report tracking the number of passing and failed tests over time.

MSTest

The MSTest integration, written as a Plugin, is added to your AnthillPro job as a Report Publisher job step. Once added to your build job, the MSTest integration enables AnthillPro to collect the unit-test report and make the findings available on the Dashboard. See View Reports (MSTest).

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbancode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

MSTest Prerequisites

- You should already have MSTest running (and producing a report on) your unit tests.
- You will need to know the location of the reports.
- You will need administrative permissions for the project.

Configure MSTestJob Step

The step should be included in your build job after MSTest has been run and the report generated. Typically, this is will be after the build step.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. Select the Insert After icon of the step that will run before the MSTest Report Publisher step.
5. Go to **Test > MSTest**, select the **MSTest Report Publisher** step, and click **Select**.
 - **Name** the step.
 - **Description**. Provide a short description.
 - **Working Directory Offset (optional)**. Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
 - **Report Name**. Give the name for this report to appear on the AnthillPro Dashboard.
 - **Base Directory**. Provide the directory for resolving MSTest XML files. Unless absolute, this is relative to the job working directory.
 - **Include Patterns**. Give the file name patterns that describe the files to be retrieved. Each include pattern must be entered on a separate line.

You can also use the following wild cards to tell AnthillPro what to include:

- ****** Indicates include every directory within the base directory.
- ***** Used to include every file. So, if you use `*.zip`, the files matching this pattern will be included.

- ****/*** Tells AnthillPro to retrieve the entire file tree underneath the base directory.
- **Exclude Patterns.** Provide the file name patterns identifying the files that will NOT be retrieved. This field is set in the same way as the Include Patterns field, only you are telling AnthillPro what NOT to include.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
- **Show Environment Variables (optional; advanced).** Give any optional environment variables in "name=value" format.

Environment variables values may contain references to existing values in the following format: `name=${env/<NAME>};value`. If the value of the `<NAME>` variable is "value2" in the current environment, then the above example will be expanded to: `name=value2;value`.

Using this technique, it is possible add an entry to PATH in the following manner: `PATH=my/path/entry;0`. Case is significant even on Windows systems.

6. Click **Save**.

View Reports (MSTest)

Once a build has completed, go to the new **Build Life** and select the **Tests** tab for information on the unit tests run. There, you can view the results and perform trending to see how your unit tests are doing over time.

E-mailing MSTest Results

Since AnthillPro is aware of MSTest, the results can be included in build notifications, just like any other piece of information. For example, you may want to automatically take action depending on results. Some common examples are to e-mail the team if a failure occurs; e-mail the QA team leader if there are no errors; or e-mail someone if the tests-per-second metric goes below a target amount.

To edit a notification template, go to the System page and choose the **Notification Templates** link from the Notification menu. See Managing Notifications.

1. Lookup the **MSTest report(s)** and put them in the **Velocity** context.
2. Once the **MSTest Reports** are available to the **Velocity** template, you can do whatever you like with them.
3. Included in an HTML-friendly e-mail template, this will produce a nice little report.

NUnit

In order to integrate with NUnit, your build scripts must run NUnit and locate the test output XML files (typically named `TestResults-*.xml`). AnthillPro takes the information from the output file and makes it available through the UI and (optionally) via e-mail.

Add NUnit Report Publisher to Build Job

Identify the NUnit output files by adding a **NUnit Report** step to the build job. This allows AnthillPro to store the data and make it available to users.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.

Before proceeding to Item Four, ensure NUnit is associated with the build script. See NUnit documentation [<http://www.nunit.org/index.php>].

4. **NUnit Report**. Select the **Insert After** icon of the step prior to the point where the NUnit step is to be included. Go to **Test > NUnit**, select the **NUnit Report** step, and click **Select**.

- **Name** the step.
- **Description**. Provide a short description.
- **Report Name**. Give the name for this report (default is same as step name) to appear on the Dashboard.
- **Source Directory**. Provide the directory where the NUnit data files will be retrieved from.
- **Include Patterns**. Give the file name patterns that describe the files to be retrieved. Each include pattern must be entered on a separate line.

You can also use the following wild cards to tell AnthillPro what to include:

- ****** Indicates include every directory within the base directory.
- ***** Used to include every file. So, if you use `*.zip`, the files matching this pattern will be included.
- ****/*** Tells AnthillPro to retrieve the entire file tree underneath the base directory.
- **Exclude Patterns**. Provide the file name patterns identifying the files that will NOT be retrieved. This field is set in the same way as the Include Patterns field, only you are telling AnthillPro what NOT to include.
- **Show Additional Options (advanced)**. Select the Show Additional Options link to configure more options.
 - **Is Active**. Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script**. From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures**. Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript**. Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.

- **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

5. Click **Save**.

View Reports (NUnit)

Once a build has completed, go to the new **Build Life** and select the **Reports** and **Tests** tabs for information on the tests run.

E-mailing NUnit Results

Since AnthillPro is aware of NUnit, the results can be included in build notifications, just like any other piece of information. For example, you may want to automatically take action depending on NUnit results. Some common examples are to e-mail the team if a failure occurs; e-mail the QA team leader if there are no errors; or e-mail someone if the tests-per-second metric goes below a target amount.

To edit a notification template, go to the System page and choose the **Notification Templates** link from the Notification menu. See Managing Notifications.

1. Lookup the **NUnit report(s)** and put them in the **Velocity** context.
2. Once the **NUnit Reports** are available to the **Velocity** template, you can do whatever you like with them.
3. Included in an HTML-friendly e-mail template, this will produce a nice little report.

Quality Center (Testing)

The Quality Center integration allows Windows users (through the COM interface) of AnthillPro to run test sets and then publish a report to the AnthillPro UI. The report is published to the Reports tab of the Build Life, and provides metrics, such as performance trends, on the tests that have been run.

The integration will only work if the agent(s) running the Quality Center steps uses the **32-bit JVM**. If the agent(s) use a 64-bit JVM, the steps will fail. However, the AnthillPro server, which does not run the steps, can run on the 64-bit JVM.

The Quality Center integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab.

Quality Center Testing Steps:

- **Run Test Set.** Runs a Test Set (QTP, LoadRunner, WinRunner) using Mercury Quality Center either directly on the server or on the Quality Center remote agent(s). Creating a separate testing workflow with the Run Test Sets and Test Set Report is advisable.
- **Publish Test Set Report.** Publishes a Quality Center TestSet Report for (QTP, LoadRunner, WinRunner). Creating a separate testing workflow with the run test sets and test set report is advisable.

The Quality Center integration also allows you to track issues with Quality Center and run QTP tests. If you want to use the QTP integration, you must configure Quality Center integration on the System page. See Quality Center Issue Tracking and QuickTest Pro.

Quality Center Prerequisites (Testing)

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- The Mercury Quality Center URL must be available.
- AnthillPro must be able to access Quality Center as a user with permissions to run tests.
- An AnthillPro agent must be installed on a Windows machine (XP, 2003, Vista, Windows 7) with the Quality Center API installed. For example, the Quality Center server. This may require configuring a Fixed Agent Filter to ensure the job runs on the appropriate machine. Alternately, the agent with the Quality Center API can be added to the Build Farm. This requires the use of a scripted filter to look for the presence of a variable/property (e.g., qc.agent=true) which must be added to the agent. If this option is chosen, it is advisable to include a filter on every build job that excludes the Quality Center agent.

The integration will only work if the agent(s) running the Quality Center steps uses the **32-bit JVM**. If the agent(s) use a 64-bit JVM, the steps will fail. However, the AnthillPro server, which does not run the steps, can run on the 64-bit JVM. See Configure and Edit Agent Filters.

- The AnthillPro agent that will perform the QualityCenter steps must be installed on the same machine as the **QualityCenter client**. You can download the client from the QualityCenter server, typically located here: `http://<qc_server>:8080/qcbin/ClientSide_index.html`. Follow the instructions on that page to install.

Configure Quality Center (Testing)

1. Go to **System > Quality Center** under the **Integration** menu.
2. On the **Mercury Quality Center Integration** page, click **Edit**.
3. Configure the integration:
 - Enter the Quality Center server URL.
 - **Issue URL**. You can have AnthillPro automatically generate a link to all of the issues it associates with a Build Life if you give the Issue URL here. If you are also using the Quality Center Issue Tracking integration, you will need to complete this field.

Once you give the URL pattern, the issues that appear on the Issues Tab of a Build Life will be linked to the issue in your issue tracker tool for reviewing the issue, adding additional comments, making edits, etc. For example, provide a URL template such as `http://bugs.company.com/browse/${issueId}`. The value `${issueId}` will be replaced in the template with the issue id of the associated issue. This field provides a template which is used throughout AnthillPro to generate links from issues directly to an issue description page within your issue tracker.

- **User Name**. Enter the user name to be used to connect to the Quality Center server. Make sure that Quality Center user AnthillPro will use to connect to the server has the appropriate permissions.
- **Password**. Enter the password to be used to connect to the Quality Center server.

- **Password Script (optional).** To use a script or property lookups for the password, leave the Password field blank and enter it here. See Scripting.

4. Click **Set** and click **Done**.

Create Quality Center Job (Testing)

Configure the job to automatically run a Quality Center test set and publish the report by setting up the job to *run on the agent with the QC COM API installed*. While each job is different, every job should run a get changelog step; run steps to interact with the changelog and Quality Center; and generate reports and make comments.

The Run Test Set and Publish Report steps are included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the typical job.

1. Go to **Administration**, select the appropriate **project**, and follow the steps for creating a build job.
2. **Quality Center - Run Test Set.** Select the **Insert After** icon of the step prior to where the step is to be included. Go to **Tests > Mercury**, select **Quality Center - Run Test Set**, and click **Select**.

If you are publishing the Test Set Report, make sure *Test Set*, *Folder*, *Domain*, and *Project* fields correspond to the Publish Test Set Report step configured below. See Quality Center - Publish Test Set Report.

- **Name.** Provide a name that will be used by AnthillPro if the default name is not used.
- **Description (optional).** Give a short description.
- **Test Set.** Give the name of the Quality Center Test Set AnthillPro will execute.
- **Folder.** Give the *full folder path* to the test set above.
- **Remote Agent.** If you want this step to run on a remote agent, give the location here. Otherwise, AnthillPro will use the local agent.
- **Domain Name.** Give the name of the Quality Center Domain where your projects are located.
- **Project Name.** Give the name of the Quality Center Project where your issues are located.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

3. Click **Save**.

4. **Quality Center - Publish Test Set Report.** Select the **Insert After** icon of the Quality Center - Run Test Set step. Go to **Test > Mercury**, select the **Quality Center - Publish Test Set Report** step, and click **Select**. This step retrieves the Report.xml files generated by the tests. Once they are retrieved from the build, AnthillPro will be able to make them available to the Build Life Tests tab.

When configuring this step, make sure *Test Set*, *Folder*, *Domain*, and *Project* corresponds to the Run Test Set step configured above. See Quality Center - Run Test Set.

- **Name** the step.
- **Description (optional).** Provide a brief description.
- **Report Name.** Provide a name for the report (if left blank, it will be the step name).
- **Test Set.** Give the name of the Quality Center Test Set AnthillPro executed in the Run Test Set step configured above.
- **Folder.** Give the *full folder path* to the test set configured in the Run Test Set step.
- **Domain Name.** Give the name of the Quality Center Domain identified in the Run Test Set step above.
- **Project Name.** Give the name of the Quality Center Project you configured in the Run Test Set step above.
- **Show Additional Options (advanced).** See Show Additional Options.

5. Click **Save**.

Add Job to Quality Center Workflow (Testing)

Complete workflow configuration is beyond the scope of this entry. The topics covered in detail below are specific to using the Quality Center integration.

1. Go **Administration** and select the appropriate workflow. You can add the job to either your build workflow or as part of a secondary (i.e., testing) workflow.
2. Go to **workflow Main > Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **job** you just created, a **job pre-condition script**, and click **Insert Job**.
6. Select the **Properties** tab and add a workflow property. This will ensure that the job runs on the appropriate agent when using a scripted agent filter. Make sure that the property you set here is also configured on the agent.
 - **Name.** Give the name of the Property (the property <name> will be accessed as `${property:<name>}`).
 - **Description.** Provide a description for this Property shown when prompting users for value.
 - **Default Value.** Input the value for this property.
 - **User May Override.** Check if users are able to specify a new value when running this workflow.
 - **Label.** Provide a label for this Property shown when prompting users for value (leave blank to use the Name as the Label).

- **Is Required.** Check if a non-empty value for this property is required to run workflow.
- **Allowed Values.** Give the values users are allowed to select for this property (blank for no restriction of value). Separate each value by entering it on its own line.
- Click **Save**.

7. Click **Save**.

Run Build and View Report (Quality Center Testing)

1. Go to **Dashboard** and select the appropriate workflow.
2. On the **workflow Main** page, click the **Build** button.
3. Once the workflow is complete, select the **Tests** tab to view the report.

Quality Center Function Calls

Following is a list of the function calls used in the Quality Center integration:

TDApi-Ole80.TDConnection	TestSet.StartExecution	TestSet.TSTestFactory	Step.Field
TDConnection.BugFactory	TSScheduler.RunAllLocally	TSTestFactory.NewList	BugFactory.Filter
TDConnection.InitConnectionEx	TSScheduler.TdHostName	TSTest.Name	BugFactory.Item
TDConnection.Login	TSScheduler.Run	TSTest.Type	BugFactory.AddItem
TDConnection.Connect	TSScheduler.ExecutionStatus	TSTest.Status	Filter.Filter
TDConnection.Disconnect	ExecutionStatus.RefreshExecStatusInfo	TSTest.HostName	Filter.NewList
TDConnection.Logout	ExecutionStatus.Finished	TSTest.TestId	Bug.ID
TDConnection.ReleaseConnection	ExecutionStatus.EventsList	TSTest.TestName	Bug.Summary
TDConnection.TestSetTreeManager	ExecEventInfo.EventType	TSTest.LastRun	Bug.Field
TestSetTreeManager.NodeByPath	ExecEventInfo.EventDate	Run.Name	Bug.AssignedTo
TestSetFolder.FindTestSets	ExecEventInfo.EventTime	Run.Status	Bug.DetectedBy
TestSet.Id	TestExecStatus.TestId	Run.Field	Bug.Priority
TestSet.Item	TestExecStatus.TestInstance	Run.StepFactory	Bug.Project
TestSet.Name	TestExecStatus.TsTestId	StepFactory.NewList	Bug.Status
TestSet.TestSetFolder	TestExecStatus.Message	Step.Name	Bug.Post
TestSet.Status	TestExecStatus.Status	Step.Status	

Quality Center Plugin (Testing)

The Quality Center integration allows Windows users (through the COM interface) of AnthillPro to run test sets and then publish a report to the AnthillPro UI. The report is published to the Reports tab of the Build Life, and provides metrics, such as performance trends, on the tests that have been run.

The integration is written as an AnthillPro Plugin, and expands upon the existing Quality Center (Testing) integration. The integration is included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbancode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

The Quality Center integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job.

The integration will only work if the agent(s) running the Quality Center steps uses the **32-bit JVM**. If the agent(s) use a 64-bit JVM, the steps will fail. However, the AnthillPro server, which does not run the steps, can run on the 64-bit JVM.

Once the job is configured, it is then added to the workflow under the Definition tab.

Quality Center Testing Steps:

- **Run Quality Center Test Set.** Runs a Test Set (QTP, LoadRunner, WinRunner) using Mercury Quality Center either directly on the server or on the Quality Center remote agent(s). Creating a separate testing workflow with the Run Test Sets and Test Set Report is advisable.
- **Publish Quality Center Test Report.** Publishes a Quality Center TestSet Report for (QTP, LoadRunner, WinRunner). Creating a separate testing workflow with the run test sets and test set report is advisable.
- The AnthillPro agent that will perform the QualityCenter steps must be installed on the same machine as the **QualityCenter client**. You can download the client from the QualityCenter server, typically located here: `http://<qc_server>:8080/qcbin/ClientSide_index.html`. Follow the instructions on that page to install.

The Quality Center integration also allows you to track issues with Quality Center and run QTP tests. If you want to use the QTP integration, you must configure Quality Center integration on the System page. See Quality Center Plugin (Issue Tracking) and QuickTest Pro.

Quality Center Plugin Prerequisites (Testing)

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- The Mercury Quality Center URL must be available.
- AnthillPro must be able to access Quality Center as a user with permissions to run tests.
- An AnthillPro agent must be installed on a Windows machine (XP, 2003, Vista, Windows 7) with the Quality Center API installed. For example, the Quality Center server. This may require configuring a Fixed Agent Filter to ensure the job runs on the appropriate machine. Alternately, the agent with the Quality Center API can be added to the Build Farm. This requires the use of a scripted filter to look for the presence of a variable/property (e.g., `qc.agent=true`) which must be added to the agent. If this option is chosen, it is advisable to include a filter on

every build job that excludes the Quality Center agent. See Configure and Edit Agent Filters.

The integration will only work if the agent(s) running the Quality Center steps uses the **32-bit JVM**. If the agent(s) use a 64-bit JVM, the steps will fail. However, the AnthillPro server, which does not run the steps, can run on the 64-bit JVM.

Configure Quality Center Plugin (Testing)

The information given here will be used by your AnthillPro projects. If you are using both the Quality Center (Testing) and Quality Center (Issue Tracking) Plugins, you need only configure the integration once (assuming you have only one Quality Center server) -- both integrations use the same System configuration.

If you are configuring integrations with multiple Quality Center servers, create a new integration for each one.

1. Go to **System > Quality Center Plugin** under the **Integration** menu.
2. On the **Quality Center Plugin** page, click **Create New**.
3. Configure the integration:
 - **Name.** Give a unique name for this integration. The name given here will be used throughout the AnthillPro system -- specifically during job creation. If you are configuring integrations with multiple Quality Center servers, ensure that each name is unique.
 - **Server URL.** Enter the base URL to the TeamForge installation base URL: `http://qualitycenter.company.com`.
 - **User Name.** Enter the user name to be used to connect to the Quality Center server. Make sure that Quality Center user AnthillPro will use to connect to the server has the appropriate permissions.
 - **Password.** Enter the password to be used to connect to the Quality Center server. If using a password script below, leave this field blank.
 - Confirm password.
 - **Password Script.** To use a script or property lookups for the password, leave the Password field blank and enter it here. See Scripting.
4. Click **Set** and click **Done**.

Create Quality Center Plugin Job (Testing)

Configure the job to automatically run a Quality Center test set and publish the report by setting up the job to *run on the agent with the QC COM API installed*. While each job is different, every job should run a get changelog step; run steps to interact with the changelog and Quality Center; and generate reports and make comments.

The Run Test Set and Publish Report steps are included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the typical job.

1. Go to **Administration**, select the appropriate **project**, and follow the steps for creating a build job.

2. **Run Quality Center Test Set.** Select the **Insert After** icon of the step prior to where the step is to be included. Go to **Test > Quality Center Plugin**, select **Run Quality Center Test Set**, and click **Select**.

If you are publishing the Test Set Report, make sure *Test Set*, *Folder*, *Domain*, and *Project* fields correspond to the Publish Test Set Report step configured below.

- **Name.** Provide a name that will be used by AnthillPro if the default name is not used.
- **Description (optional).** Give a short description.
- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **QC Server.** Select the correct integration from the drop-down menu (this is the integration set up in the Configure Quality Center Plugin (Testing) section). If you configured multiple integrations on the AnthillPro System page, make sure you select the correct one. Note that it is possible for a single job -- but not a step -- to use different AnthillPro/Quality-Center-server configurations.
- **Domain Name.** Give the name of the Quality Center Domain where your projects are located.
- **Project Name.** Give the name of the Quality Center Project where your issues are located.
- **Folder.** Give the *full folder path* to the test set above.
- **Test Set.** Give the name of the Quality Center Test Set AnthillPro will execute.
- **Remote Host.** If you want this step to run on a remote host (i.e., a remote agent), give the location here. Otherwise, AnthillPro will use the agent on the local host.
- **Show Environment Variables (optional; advanced).** Give any optional environment variables in "name=value" format.

Environment variables values may contain references to existing values in the following format: `name=${env/<NAME>} ;value`. If the value of the `<NAME>` variable is "value2" in the current environment, then the above example will be expanded to: `name=value2 ;value`.

Using this technique, it is possible add an entry to PATH in the following manner: `PATH=my/path/entry ; 0`. Case is significant even on Windows systems.

- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

3. Click **Save**.

4. **Publish Quality Center Test Report.** Select the **Insert After** icon of the previous step. Go to **Test > Quality**

Center Plugin, select the **Publish Quality Center Test Report** step, and click **Select**. This step retrieves the Report.xml files generated by the tests. Once they are retrieved from the build, AnthillPro will be able to make them available to the Build Life Tests tab.

When configuring this step, make sure *Test Set*, *Folder*, *Domain*, and *Project* corresponds to the Run Test Set step configured above.

- **Name** the step.
- **Description (optional)**. Provide a brief description.
- **Working Directory Offset (optional)**. Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **QC Server**. Select the correct integration from the drop-down menu (this is the integration set up in the Configure Quality Center Plugin (Testing) section). If you configured multiple integrations on the AnthillPro System page, make sure you select the correct one. Note that it is possible for a single job -- but not a step -- to use different AnthillPro/Quality-Center-server configurations.
- **Domain Name**. Give the name of the Quality Center Domain identified in the Run Test Set step above.
- **Project Name**. Give the name of the Quality Center Project you configured in the Run Test Set step above.
- **Folder**. Give the *full folder path* to the test set configured in the Run Test Set step.
- **Test Set**. Give the name of the Quality Center Test Set AnthillPro executed in the Run Test Set step configured above.
- **Show Environment Variables (optional; advanced)**. See Show Environment Variables above.
- **Show Additional Options (advanced)**. See Show Additional Options.

5. Click **Save**.

Add Job to Quality Center Plugin Workflow (Testing)

Complete workflow configuration is beyond the scope of this entry. The topics covered in detail below are specific to using the Quality Center integration.

1. Go **Administration** and select the appropriate workflow. You can add the job to either your build workflow or as part of a secondary (i.e., testing) workflow.
2. Go to **workflow Main > Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **job** you just created, a **job pre-condition script**, and click **Insert Job**.
6. Select the **Properties** tab and add a workflow property. This will ensure that the job runs on the appropriate agent when using a scripted agent filter. Make sure that the property you set here is also configured on the agent.
 - **Name**. Give the name of the Property (the property <name> will be accessed as `${property:<name>}`).

- **Description.** Provide a description for this Property shown when prompting users for value.
- **Default Value.** Input the value for this property.
- **User May Override.** Check if users are able to specify a new value when running this workflow.
- **Label.** Provide a label for this Property shown when prompting users for value (leave blank to use the Name as the Label).
- **Is Required.** Check if a non-empty value for this property is required to run workflow.
- **Allowed Values.** Give the values users are allowed to select for this property (blank for no restriction of value). Separate each value by entering it on its own line.
- Click **Save**.

7. Click **Save**.

Run Build and View Report (Quality Center Plugin Testing)

1. Go to **Dashboard** and select the appropriate workflow.
2. On the **workflow Main** page, click the **Build** button.
3. Once the workflow is complete, select the **Tests** tab to view the report.

Quality Center Plugin Function Calls

Following is a list of the function calls used in the Quality Center integration:

TDApi-Ole80.TDConnection	TestSet.StartExecution	TestSet.TSTestFactory	Step.Field
TDConnection.BugFactory	TSScheduler.RunAllLocally	TSTestFactory.NewList	BugFactory.Filter
TDConnection.InitConnectionEx	TSScheduler.TdHostName	TSTest.Name	BugFactory.Item
TDConnection.Login	TSScheduler.Run	TSTest.Type	BugFactory.AddItem
TDConnection.Connect	TSScheduler.ExecutionStatus	TSTest.Status	Filter.Filter
TDConnection.Disconnect	ExecutionStatus.RefreshExecStatusInfo	TSTest.HostName	Filter.NewList
TDConnection.Logout	ExecutionStatus.Finished	TSTest.TestId	Bug.ID
TDConnection.ReleaseConnection	ExecutionStatus.EventsList	TSTest.TestName	Bug.Summary
TDConnection.TestSetTreeManager	ExecEventInfo.EventType	TSTest.LastRun	Bug.Field
TestSetTreeMan-	ExecEventInfo.EventDate	Run.Name	Bug.AssignedTo

ager.NodeByPath			
TestSetFolder.FindTestSets	ExecEventInfo.EventTime	Run.Status	Bug.DetectedBy
TestSet.Id	TestExecStatus.TestId	Run.Field	Bug.Priority
TestSet.Item	TestExec-Status.TestInstance	Run.StepFactory	Bug.Project
TestSet.Name	TestExecStatus.TsTestId	StepFactory.NewList	Bug.Status
TestSet.TestSetFolder	TestExecStatus.Message	Step.Name	Bug.Post
TestSet.Status	TestExecStatus.Status	Step.Status	

QuickTest Pro

Running tests with the QTP integration, it is possible to publish the Test Directory to store the tests in the SCM; publish the tests to AnthillPro during the build; then resolve the tests during the deployment to be run (on a machine with QTP) after an instance of the application has been deployed/setup/installed.

The integration will only work if the agent(s) running the QuickTest Pro steps uses the **32-bit JVM**. If the agent(s) use a 64-bit JVM, the steps will fail. However, the AnthillPro server, which does not run the steps, can run on the 64-bit JVM.

The QuickTest Pro integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab.

QTP Steps:

- **QuickTest Pro - Run Tests.** Runs a set of tests in the Mercury QuickTest Pro tool.
- **QuickTest Pro - Publish Report.** Publishes a QuickTest Pro Report.

The QuickTest Pro integration can be used in conjunction with Quality Center. You must configure the Quality Center integration on the System page. See [Quality Center Testing](#) and [Quality Center Issue Tracking](#).

QTP Prerequisites

- The Quality Center integration must be configured on the system page.
- You must have administrative privileges to create workflows. See [Manage Security](#).
- A Fixed Agent Filter that runs on the agent with the QC COM API installed.

Alternately, the agent with the Quality Center API can be added to the Build Farm. This requires the use of a scripted filter to look for the presence of a variable/property (e.g., qc.agent=true) which must be added to the agent. If this option is chosen, it is advisable to include a filter on every build job that excludes the Quality Center agent. See [Configure and Edit Agent Filters](#).

The integration will only work if the agent(s) running the QuickTest Pro steps uses the **32-bit JVM**. If the agent(s) use a 64-bit JVM, the steps will fail. However, the AnthillPro server, which does not run the steps, can run on the 64-bit JVM.

Configure Run QTP Test Job

1. Go to **Administration**, select the **project** that is to be tested, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. On the next page, **name** the job, give a **description (optional)** and click **Set**.
4. On the **job configuration** page, click **Create Step**.
5. **QuickTest Pro - Run Tests**. Select the **Insert After** icon of the step prior to where the QuickTest Pro - Run Tests step is to be included (typically after the Set Working Directory step). Go to **Tests > Mercury**, select **QuickTest Pro - Run Tests** step, and click **Select**.
 - **Name**. Provide a name.
 - **Description (optional)**. Give a short description.
 - **Base Test Path**. Provide the directory containing all QuickTest Pro tests to be run.
 - **Web Application URL**. If testing a web application, this field will override the application URL.
 - **Result Base Path**. To override the default result output location for the tests, give the new path. A new directory will be created for each test under the specified path.
 - **Browser**. Give the browser being used during testing. In some cases, the browser must be specified in order for the test to run successfully.
 - **Fail On Error**. Check the box to fail the step if any errors occur during the test.
 - **Fail On Warning**. Check the box to fail the step if any warnings occur during the test.
 - **Show Additional Options (advanced)**. See Show Additional Options.
6. Click **Save**.
7. **QuickTest Pro - Publish Report**. Select the **Insert After** icon of the QuickTest Pro - Run Tests step. Go to **Test > Mercury**, select the **QuickTest Pro - Publish Report** step, and click **Select**. This step retrieves the Report.xml files generated by QuickTest Pro. Once they are retrieved from the build, AnthillPro will be able to access the results in a meaningful way.
 - **Name** the step.
 - **Description (optional)**. Provide a brief description.
 - **Report Name**. Provide a name for the report (if left blank, it will be the step name).
 - **Source Directory**. The directory where the QuickTest Pro data files will be retrieved from.
 - **Include Patterns**. File name patterns that describe the files that will be retrieved.
 - **Excluded Patterns**. File name patterns identifying the files that will NOT be retrieved.
 - **Show Additional Options (advanced)**. See Show Additional Options.
8. Click **Save**.

Configure the Run QTP Tests Workflow

Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the QuickTest Pro integration.

1. Go **Administration** and select the appropriate workflow. You can add the job to either your build workflow or as part of a secondary (i.e., testing) workflow.
2. Go to **workflow Main > Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the job created in Configure Run QTP Test Job of this tutorial, a **job pre-condition script**, and click **Insert Job**.
6. The Job will appear on the workflow definition page.

Run QTP Workflow and Get Reports

1. Go to **Dashboard > Build Life** to be tested.
2. On the **Build Life** page, click the **Run Secondary Process** button.
3. From the drop-down box, choose the workflow to be run, and click **Next**.
4. Select the **environment** and click the **Run** button.
5. Once the workflow is complete, go to the **Build Life** page and select the **Reports** tab.

In addition to publishing the test results on the Build Life page, AnthillPro also stores the results in its data warehouse, thus enabling you to compare your test runs over time (trending).

6. Select the **Publish_QTP_Results** link to view the **QuickTest Pro** report.
7. To drill down on each Quality Center step on the Build Life Summary page, see Trace a Build Life to Source.

QuickTest Pro Function Calls

Following is a list of the function calls used in the Quality Center integration:

QuickTest.Application	Test.Run
Application.Open	Test.IsRunning
Application.Quit	Settings.Launchers
Application.Test	Launchers.Item
Test.Settings	Launchers.Item("Web").Address

Selenium

Use the Selenium integration to run any Selenium test suite. Selenium may be integrated as part of the build job or used as part of a non-originating workflow (secondary process) on any Build Life. Once the tests have been run, metrics are available on the Build Life Reports and Tests tabs.

How the integration is used will typically be determined by (a.) the duration of the test suite to be run; and (b.) when the test suite is to be run. For most users, running Selenium as part of a non-originating workflow (secondary process) is advisable. For example, to automatically run the test suite every time a build is performed, add a 'Run Another Workflow' job to the build (make sure the second job does not start until the build is complete). This will tell AnthillPro to run the secondary workflow that kicks off the test suite. Or, you can set up a manual task that runs the Selenium workflow (see Execute Workflow via Task). It is also possible to create a schedule that runs the test suite by creating a scheduled trigger on the workflow (see Use Triggers and Scheduled Builds).

The integration is implemented as an AnthillPro job step configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add the step to a job. Once the job is configured, it is then added to the workflow under the Definition tab.

Selenium Step:

Run Tests. Runs a Selenium test suite. May be used as part of an originating and/or non-originating workflow. See Using the Selenium Integration.

Selenium Prerequisites

- Selenium must be active, and access available to AnthillPro. See Selenium documentation [<http://seleniumhq.org/documentation/>].
- You must have access to the Administration page. See Manage Security.
- At least one Build Life must be active.

Using the Selenium Integration

The Selenium integration allows AnthillPro users to run a Selenium test suite by configuring a non-originating workflow (secondary process) to be run on any successful Build Life. This section only covers the steps necessary for creating a job that uses Selenium and a secondary workflow that kicks off the test suite. While your job configuration will vary, the Selenium integration will most likely use a Set Working Directory and Run Selenium Test step in the secondary workflow (for other options, see Selenium).

1. Create a secondary workflow that will run the test suite.
2. Create a job to be added to the newly created workflow.
3. On the job page, click the Create Step button to add a Set Working Directory step to the job.
4. Select the Insert After icon of the Set Working Directory step. Go to **Test > Selenium**, select Run Tests, and click **Select**.
5. Configure Step:
 - **Name** the step.
 - **Description.** Provide a description.
 - **Test Suite Path.** Give the path to the Selenium test suite to be run. This is relative to the working directory.

- **Web Application URL.** If you need to override the application's base URL, give it here.
- **Browser.** Give the Selenium browser designation. Tests can only be run in one browser at a time. To run tests in multiple browsers, create a new step for each browser type.
- **Results file.** Give the name of the file that contains the Selenium test results.
- **Port.** If Selenium is using a port other than the default (4444), give it here. Otherwise, leave this field blank.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

6. Click **Save**.

7. See Get Selenium Reports.

Get Selenium Reports

1. Go to the **Dashboard** and select the originating workflow to be tested.
2. On the **Main** page, select the appropriate **Build Life**.
3. On the **Build Life** Main page, click the **Run Secondary Process** button.
4. From the drop-down box, choose the workflow created in the Using the Selenium Integration section, and click **Next**.
5. Select the **environment** and click the **Run** button.
6. Select the test from the **Publish Reports** or **Tests** menu.

SilkCentral

Use the SilkCentral integration to run any test suite managed by Borland's SilkCentral Test Manager (see Using the SilkCentral Integration). SilkCentral may be integrated as part of the build job or used as part of a non-originating workflow (secondary process) on any Build Life. Once the tests have been run, metrics are available on the Build Life Reports and Tests tabs (see also Run SilkCentral Build Workflow and Run SilkCentral Secondary Workflow).

How the integration is used will typically be determined by (a.) the type and duration of the test suite to be run; and (b.) when the test suite is to be run. For example, to run Unit (or other short duration) tests as part of a Continuous Integration build, the Run Tests step must be included as part of the build process. See Use SilkCentral as Part of a

Build.

The integration may also be used to run long tests, such as functional tests, as part of a secondary workflow that does not require any other job steps. See [Use SilkCentral as Part of a Secondary Workflow](#).

In order to use the integration, AnthillPro must first be configured with SilkCentral (see [Set Up SilkCentral](#)). The integration is implemented as an AnthillPro job step configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add the step to a job. Once the job is configured, it is then added to the workflow under the Definition tab.

SilkCentral Step:

Run Tests. Runs a test suite using Borland SilkCentral Test Manager. May be used as part of an originating and/or non-originating workflow.

This tutorial will follow a simple project configuration that first uses the SilkCentral Run Tests step as part of an originating workflow (i.e., a Continuous Integration build job) to run Unit tests, and as a secondary workflow to run functional tests, etc. The example in this tutorial uses Subversion, but the basic configuration is similar for any repository type. Your jobs will vary, but the SilkCentral integration is added as a job step similar to what is described below.

Set Up SilkCentral

Once set up as a SilkCentral user, AnthillPro must be able to access the server in order for the integration to work. Any steps relying on the SilkCentral integration will not work until the configuration is complete.

All fields may contain scripts and/or property lookups. See [Scripting](#).

SilkCentral Prerequisites

- You must have AnthillPro administrative privileges. See [Manage Security](#).
- A project must be active in AnthillPro.
- The SilkCentral URL must be available.
- AnthillPro must be set up as a SilkCentral user. See [SilkCentral documentation](#).

Configure SilkCentral

1. Go to **System > SilkCentral Test Manager** from the Integrations menu.
2. On the **Borland SilkCentral Test Manager Integration** page, click **Edit**.
3. Configure the integration:
 - **Borland SilkCentral Server URL.** Provide the SilkCentral server URL.
 - **User Name.** Give the SilkCentral user name assigned to AnthillPro.
 - **Password.** Provide the password AnthillPro will to connect to the SilkCentral server.
 - **Password Script.** To use a script or property lookups for the password, leave the Password field blank and enter the script here. See [Scripting](#).
4. Click **Set** then **Done**.

Using the SilkCentral Integration

The SilkCentral integration allows AnthillPro users to automate the test orchestration process by configuring a build (originating) workflow to run SilkCentral-managed tests during the build, or as a non-originating workflow (secondary process) to be run on any successful Build Life. This section only covers the steps necessary for creating a build job that uses SilkCentral and a secondary workflow that kicks off the test manager. While your job configuration will vary, the SilkCentral integration is used similar to what is described below.

Using the SilkCentral Integration Prerequisites

- The Set Up SilkCentral section of this tutorial must be complete.
- You must have administrative privileges. See Manage Security.
- A project with at least one Build Life must be active in AnthillPro (if using the integration as part of a secondary workflow).

Use SilkCentral as Part of a Build

To run Unit (or other short duration) tests as part of a Continuous Integration build, the Run Tests step must be included as part of the build process that typically includes a set working directory, get changelog, and publish changelog step. Once the SilkCentral Run Tests step is added to the build job, AnthillPro will kick off the tests associated with the step as part of the build, and provide feedback on the Dashboard Build Life page.

The items below only cover the steps necessary to using the SilkCentral integration as part of a build workflow.

Configure SilkCentral Build Job

Each build job is different; however the SilkCentral step is configured similar to what is below. It is also possible to configure AnthillPro to make decisions to pass or fail the build based on the results of the tests run during the build.

1. Go to **Administration**, select the **project** that is to be tested, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. On the next page, **name** the job, give a **description (optional)** and click **Set**.
4. Follow the steps for creating a job.
5. **Run Tests.** Click the **Insert After** button of the step prior to the point where the Run Tests step is to be included (e.g., typically after the set working directory, build, get changelog steps). Go to **Test > Silk Central Test Manager**, select **Run Tests** step, and click **Select**.
 - **Name** the step.
 - **Description.** Provide an optional description.
 - **Project Name.** Give the name of the SilkCentral project.
 - **Tests.** Provide the path, relative to the SilkCentral project, of all the tests to be run by this job. If the tests are to be identified by scripts, separate them by a comma; if the path is hard coded, each test can be input on a separate line.
 - **Ignore Missing Tests.** Check the box to have AnthillPro ignore any missing tests. If a test is defined in the Tests field above and is missing, AnthillPro will fail the job unless the box is checked.

- **Execution Server Host.** To restrict the host on which the tests can run, give the location of the SilkCentral Execution Host. If left blank, AnthillPro will run the tests on any of the available hosts.
- **Execution Server Port.** If the execution server is restricted to a particular port, give it here. Inputting a value of -1 (negative one) will allow the tests to run on any available port.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

6. Click **Save**.

Configure SilkCentral Build Workflow

The SilkCentral job must be executed as part of a build workflow in order to run Unit tests as part of the Continuous Integration build. This section assumes familiarity with the originating workflow creation process, and only covers the topics necessary to adding the job (see Configure SilkCentral Build Job) to the workflow.

1. Go to **Administration** and select the **Add Workflow** icon of the appropriate project.
2. Check **Originating Workflow** and click **Select**.
3. Configure the workflow.
4. On the workflow **Main** page, select the **Definition** tab.
5. From the drop-down menu, choose **Embedded Definition** and click **Select**.
6. Left-click the **Start** icon and select **Insert Job After**.
7. Select the **Build** job created in the Configure SilkCentral Build Job section, a **job pre-condition** script, and click **Insert Job**.

Run Build Workflow and Get Reports (SilkCentral)

1. Go to **Dashboard** and select the workflow created in the Configure SilkCentral Build Workflow section.
2. On the workflow **Main** page, click the **Build** button.
3. Once the workflow is complete, go to the **Build Life** page and select the **Reports** tab.
4. Select the test from the **Publish Reports** menu. A detail of each test run (not shown) is also available.

\$report.testType Report

Test Summary:

Total:	Pass:	Fail:	Success Rate:	Time:
5	3	0	60%	36.2 secs

Class Summary:

Name:	Tests:	Failures:	Time:
/IE6_Related	2	0	17.2 secs
/FireFox_Related	2	0	2.3 secs
test folder/test definition	1	0	16.8 secs

[Back to Top](#)

- Once the workflow is complete, go to the **Build Life** page and select the **Tests** tab.

The screenshot shows the 'Tests' tab in the SilkCentral interface. At the top, there are navigation tabs: Main, Reports, Artifacts, Dependencies, Changes, Tests (selected), Issues, and Notes. A version indicator shows '10.18 | 10.20'. Below the tabs, the 'Test Reports: [1]' section is visible. It contains a summary for a test run: 'Name: Run Silk Central Test Manager (published from job 93)', 'Type: # of Suites: 3', and 'Test Results: 3 / 5 success (60%)'. A 3D pie chart visualizes the results, with a green slice representing the 60% success rate and a grey slice representing the 40% failure rate. There is also a 'Hide Test Suites' checkbox.

- To drill down on each step on the Build Life Summary page, see Trace a Build Life to Source.

Use SilkCentral as Part of a Secondary Workflow

To automatically kick off long tests (e.g., functional tests), the SilkCentral Run Tests step is typically configured as part of a secondary workflow which is run on an existing Build Life. The integration does not require any other job steps to be run as part of the workflow, and, for example, can be scheduled to kick off the test suite using a schedule or trigger (see Triggers and Scheduled Builds).

The items below only cover the steps necessary to using the SilkCentral integration as part of a secondary (non-originating) workflow.

Configure SilkCentral Secondary Job

The Run Test step may be used on its own; however, depending on the needs of each project, the job may require other steps to start a virtual machine, pass links, evaluate scripts, etc.

- Go to **Administration**, select the **project** that is to be tested, and click the **Add Job** icon.
- On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
- On the next page, **name** the job, give a **description (optional)** and click **Set**.

4. Follow the steps for creating a job.
5. **Run Tests.** Click the **Create Step** button. Go to **Test > Silk Central Test Manager**, select **Run Tests** step, and click **Select**.
 - **Name** the step.
 - **Description.** Provide an optional description.
 - **Project Name.** Give the name of the SilkCentral project.
 - **Tests.** Provide the path, relative to the SilkCentral project, of all the tests to be run by this job. If the tests are to be identified by scripts, separate them by a comma; if the path is hard coded, each test can be input on a separate line.
 - **Ignore Missing Tests.** Check the box to have AnthillPro ignore any missing tests. If a test is defined in the Tests field above and is missing, AnthillPro will fail the job unless the box is checked.
 - **Execution Server Host.** To restrict the host on which the tests can run, give the location of the SilkCentral Execution Host. If left blank, AnthillPro will run the tests on any of the available hosts.
 - **Execution Server Port.** If the execution server is restricted to a particular port, give it here. Inputting a value of -1 (negative one) will allow the tests to run on any available port.
 - **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
6. Click **Save**.

Configure SilkCentral Secondary Workflow

The SilkCentral job can be executed as part of a secondary (non-originating) workflow in order to run functional (or other) tests on an existing Build Life. While there are numerous ways to use the SilkCentral integration to automatically run testing suites, your particular workflow will be similar to what is presented here. For example, you can set up the workflow to run on a scheduled trigger (see Triggers and Scheduled Builds) that kicks off the testing suite, or can even include functional and other long running testing as part of the build process (see also Use SilkCentral as Part of a Build).

1. Go to **Administration** and select the **Add Workflow** icon of the appropriate project.
2. Check **Non-originating Workflow** and click **Select**.
3. Configure workflow.

4. On the workflow **Main** page, select the **Definition** tab.
5. From the drop-down menu, choose **Embedded Definition** and click **Select**.
6. Left-click the **Start** icon and select **Insert Job After**.
7. Select the **Build** job created in the Configure SilkCentral Secondary Job section, a **job pre-condition** script, and click **Insert Job**.
8. If setting a trigger, see Triggers and Scheduled Builds.

Run Secondary Workflows and Get Reports (SilkCentral)

1. Go to **Dashboard** and select the originating workflow to be tested.
2. On the **Main** page, select the appropriate **Build Life**.
3. On the **Build Life** Main page, click the **Run Secondary Process** button.
4. From the drop-down box, choose the workflow created in the Configure SilkCentral Secondary Workflow section, and click **Next**.
5. Select the **environment** and click the **Run** button.
6. Once the workflow is complete, go to the **Build Life** page and select the **Reports** tab.
7. Select the test from the **Publish Reports** menu. A detail of each test run (not shown) is also available.

\$report.testType Report

Test Summary:

Total:	Pass:	Fail:	Success Rate:	Time:
5	3	0	60%	36.2 secs

Class Summary:

Name:	Tests:	Failures:	Time:
/IE6_Related	2	0	17.2 secs
/FireFox_Related	2	0	2.3 secs
test folder/test definition	1	0	16.8 secs

[Back to Top](#)

8. Once the workflow is complete, go to the **Build Life** page and select the **Tests** tab.



9. To drill down on each step on the Build Life Summary page, see Trace a Build Life to Source.

TestNG

To use TestNG with AnthillPro, publish the results in the JUnit format (See TestNG documentation [<http://testng.org/doc/documentation-main.html>]) and configure the JUnit Integration. Once TestNG conforms to the JUnit output AnthillPro expects, the reports can then be made available in AnthillPro or sent in a notification -- exactly like any other JUnit report (see the JUnit integration for more information).

AnthillPro has an integration with TestNG that publishes the test data to the Build Life Test tab. The integration consists of a single job step that is added near the end of your job.

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbancode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

Once you have AnthillPro running your TestNG tests, you can add the **TestNG Publish Test Report** step near the end of your job:

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a job that runs your tests.
4. Select the Insert After icon of the step that will run before the TestNG Report Publisher step.
5. Go to **Test > TestNG**, select the **TestNG Report Publisher** step, and click **Select**.
 - **Name** the step.
 - **Description**. Provide a short description.
 - **Working Directory Offset (optional)**. Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
 - **Report Name**. Give the name for this report to appear on the AnthillPro Dashboard. If none is given, TestNG will be used.
 - **Base Directory**. Provide the directory for resolving TestNG XML files. Unless absolute, this is relative to the job working directory.

- **Include Patterns.** Give the file name patterns that describe the files to be retrieved. Each include pattern must be entered on a separate line.

You can also use the following wild cards to tell AnthillPro what to include:

- ****** Indicates include every directory within the base directory.
- ***** Used to include every file. So, if you use `*.zip`, the files matching this pattern will be included.
- ****/*** Tells AnthillPro to retrieve the entire file tree underneath the base directory.
- **Exclude Patterns.** Provide the file name patterns identifying the files that will NOT be retrieved. This field is set in the same way as the Include Patterns field, only you are telling AnthillPro what NOT to include.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
- **Show Environment Variables (optional; advanced).** Give any optional environment variables in "name=value" format.

Environment variables values may contain references to existing values in the following format: `name=${env/<NAME>} ; value`. If the value of the `<NAME>` variable is "value2" in the current environment, then the above example will be expanded to: `name=value2 ; value`.

Using this technique, it is possible add an entry to PATH in the following manner: `PATH=my/path/entry ; 0`. Case is significant even on Windows systems.

6. Click **Save**.
7. Once the build is run, go to the new Build Life and select the Test tab. There, you can investigate the findings.

Chapter 66. Coverage Tools

The coverage integrations, implemented as job steps, allow AnthillPro to collect information on the coverage tests that you invoke with your build script. Once configured, AnthillPro will collect the output and store it in the AnthillPro data warehouse. This makes it possible for you to perform trending over time, see which tests are failing, track coverage, and even fail a build based on the captured results.

Each integration allows you to add the following job steps as part of your build process:

- **Retrieve the coverage report.** AnthillPro gets any reports generated by the coverage tool and publishes them on the build's Dashboard.
- **Fail a build based on coverage.** Using a script, you can have AnthillPro fail a build if the percentage of test coverage falls below a preset value.

Clover

Run Clover and publish coverage reports with the Clover 1.x integration. Users can also fail a workflow based on percentage of code coverage (see [Configure Evaluate Script Step \[Clover\]](#)).

To use the integration, Clover is added to the builder either through the command line, as an Ant task, or as a Plugin within the Maven environment (see [Codestation \[Developers\]](#) and [Clover documentation \[http://confluence.atlassian.com/display/CLOVER/Clover+Documentation+Home\]](http://confluence.atlassian.com/display/CLOVER/Clover+Documentation+Home)). Once Clover has been added to the builder, configure the report publisher as a step during job configuration.

Clover steps:

- **Clover Coverage Report.** Retrieves the report generated by Clover for the individual build.
- **Evaluate Script.** Reads the information in the database and then fails a workflow if the percentage of test coverage is less than the minimum limit.

Your jobs will vary, but the Clover integration is added as a job step similar to what is described below. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to a workflow (under the Definition tab).

The Clover job will be typically configured to make the coverage report available to the AnthillPro UI via the Build Life Reports tab. Once Clover is configured with the builder, the Clover Report step is included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the typical Build job.

Clover Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See [Manage Security](#).
- A project with at least one Build Life must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See [Using Life-Cycle Models](#).
- Clover must be added to your command-line build script, as an Ant task, or as part of the Maven environment.

See [Codestation \(Developers\)](http://confluence.atlassian.com/display/CLOVER/Clover+Documentation+Home) and [Clover documentation](http://confluence.atlassian.com/display/CLOVER/Clover+Documentation+Home) [http://confluence.atlassian.com/display/CLOVER/Clover+Documentation+Home].

- The source directory where the Clover report file is located must be available.

Configure Clover Report Step

Configure the Clover Report step in this section. This step is tasked with retrieving the report generated by Clover.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.

Before proceeding to Item Four, add Clover to your build script, as an Ant task, or as part of the Maven environment. See [Clover documentation](http://confluence.atlassian.com/display/CLOVER/Clover+Documentation+Home) [http://confluence.atlassian.com/display/CLOVER/Clover+Documentation+Home].

4. **Clover Coverage Report.** Select the **Insert After** icon of the step prior to the point where the Clover step is to be included (e.g., the Artifact Delivery step). Go to **Coverage > Clover**, select the **Clover Coverage Report** step, and click **Select**.
 - **Name** the step (required).
 - **Description.** Provide a short description.
 - **Report Name (required).** Give the name for this report (default is same as step name).
 - **Performing Digest.** Check yes to perform a digest on the published files integrity. The digest algorithm is set by the administrator in the Server Settings. The digest will not occur if no algorithm is selected. See Configure Server Security.
 - **Source Directory.** Provide the directory where the Clover report files are retrieved from.
 - **Include Patterns.** Give the file name patterns that describe the files to be retrieved. Each include pattern must be entered on a separate line.

You can also use the following wild cards to tell AnthillPro what to include:

- ****** Indicates include every directory within the base directory.
- ***** Used to include every file. So, if you use `*.zip`, the files matching this pattern will be included.
- ****/*** Tells AnthillPro to retrieve the entire file tree underneath the base directory.
- **Exclude Patterns.** Provide the file name patterns identifying the files that will NOT be retrieved. This field is set in the same way as the Include Patterns field, only you are telling AnthillPro what NOT to include.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the

status determination of the job.

- **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
- **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

5. Click **Save**.

6. If configuring the **Evaluate Script** step to fail the workflow based on coverage percentage, see Configure Evaluate Script Step (Clover). Otherwise, proceed to Add Clover Job to Workflow.

Configure Evaluate Script Step (Clover)

To have AnthillPro fail a workflow based on the Clover report, add an Evaluate Script step to the Build job. The Evaluate Script step, added after the Clover Report step, will read the information in the database and then fail a workflow if the percentage of test coverage is less than the minimum limit. See **Tools > Developer Tools > Scripting API > CoverageHelper**.

Example Coverage Report Script:

```
import com.urbancode.anthill3.domain.coverage.CoverageReport;
import org.apache.log4j.Logger;
```

```
static private final Logger log = Logger.getLogger("Stuff");
CoverageReport[] reports = null;
try {
    reports = CoverageHelper.getForCurrentBuildLife();
}
catch (Exception e) {
    log.warn(e);
}

if (reports != null) {
    log.warn("Coverage getForCurrentBuildLife " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}
```

```
CoverageReport[] reports = CoverageHelper.getForBuildLife(BuildLifeLookup.getCurrent());
if (reports != null) {
    log.warn("Coverage getForBuildLife " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}
```

```
CoverageReport[] reports = CoverageHelper.getForCurrentJobTrace();
if (reports != null) {
    log.warn("Coverage getForCurrentJobTrace " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
}
```

```
    log.warn("Coverage " + reports[0].getName());
}

CoverageReport[] reports = CoverageHelper.getForJobTrace(JobTraceLookup.getCurrent());
if (reports != null) {
    log.warn("Coverage getForJobTrace " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}

CoverageReport[] reports = CoverageHelper.getRecentForCurrentWorkflow();
if (reports != null) {
    log.warn("Coverage getRecentForCurrentWorkflow " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}

CoverageReport[] reports = CoverageHelper.getRecentForWorkflow(WorkflowLookup.getCurrent());
if (reports != null) {
    log.warn("Coverage getRecentForWorkflow " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}
```

To configure the Evaluate Script step:

1. Select the **Insert After** icon of the Clover Report step. Expand the **Miscellaneous folder**, select the **Evaluate Script** step, and click **Select**.
2. **Name** the step (required).
3. **Description**. Provide a short description.
4. **Script**. Give the BeanShell script to evaluate. See Scripting.
5. **Show Additional Options (advanced)**. Select the Show Additional Options link to configure more options.
 - **Is Active**. Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script**. From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures**. Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript**. Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout**. Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
6. Click **Save**.

Add Clover Job to Workflow

The Job with the Clover step(s) must be executed as part of a workflow (see Configure Clover Report Step). This section will assume that an originating workflow has already been configured, and will cover the process of adding the Clover build job to the appropriate workflow. Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the Clover integration.

1. Go to **Administration**, select the **project**, and select the **build** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Build** job created in the Configure Clover Report section, choose a **job pre-condition** script, and click **Insert Job**.

Run Workflow and View Report (Clover)

1. Go to **Dashboard** and select the **workflow** created in the Add Clover Job to Workflow section.
2. On the workflow **Main** page, click the **Build** button for the workflow.
3. Once the **workflow** has completed, select the appropriate **Build Life** and click the **Reports** tab.
4. Select the **Clover** link to view the Clover coverage report for this build.

Clover coverage report - xPetStore Test Coverage
 Coverage timestamp: Thu May 10 2007 15:05:27 EDT

project stats: LOC: 4,169 Methods: 324
 NCLOC: 1,941 Classes: 42
 Files: 42 Packages: 12

Project	Conditionals	Statements	Methods	TOTAL
Project	0%	6%	41.4%	19.1%
Packages	Conditionals	Statements	Methods	TOTAL
xpetstore.util	-	0%	0%	0%
xpetstore.web.filter	0%	0%	0%	0%
xpetstore.web.servlet	-	0%	0%	0%
xpetstore.web.webwork.action	0%	0%	0%	0%
xpetstore.web.webwork.action.cart	0%	0%	0%	0%
xpetstore.web.webwork.action.category	0%	0%	0%	0%
xpetstore.web.webwork.action.customer	0%	0%	0%	0%
xpetstore.web.webwork.action.item	-	0%	0%	0%
xpetstore.web.webwork.action.order	0%	0%	0%	0%
xpetstore.web.webwork.action.product	0%	0%	0%	0%
xpetstore.web.webwork.action.signon	0%	0%	0%	0%
xpetstore.domain	0%	16.9%	55.1%	39.2%

Report generated by Clover Code Coverage v1.3.1.3
 Thu May 10 2007 15:05:29 EDT. 5 Seat Team License registered to mbz@urbancode.com, Urbancode, Inc

Cobertura

Run Cobertura and publish coverage reports with the Cobertura 1.9 integration. In addition, AnthillPro users can generate a coverage report for every project that uses Cobertura. Users can also fail a workflow based on percentage

of code coverage (see Configure Evaluate Script Step [Cobertura]).

To use the integration, Cobertura is added to the builder either through the command line, as an Ant task, or as a Plugin within the Maven environment (see Codestation [Developers] and Cobertura documentation [<http://cobertura.sourceforge.net/>]). Once Cobertura has been added to the builder, configure the report publisher as a step during job configuration.

Cobertura steps:

- **Cobertura Coverage Report.** Retrieves the report generated by Cobertura for the individual build.
- **Evaluate Script.** Reads the information in the database and then fails a workflow if the percentage of test coverage is less than the minimum limit.

Your jobs will vary, but the Cobertura integration is added as a job step similar to what is described below. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to a workflow (under the Definition tab).

The Cobertura job will typically be configured to make the coverage report available to the AnthillPro UI via the Build Life Reports tab. If the coverage.xml file is stored in AnthillPro's data warehouse, metrics and trending are available on the Coverage tab. Once Cobertura is configured with the builder, the Report Publish step is included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the typical Build job.

Cobertura Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- A project with at least one Build Life must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See Using Life-Cycle Models.
- Cobertura must be added to your command-line build script, as an Ant task, or as part of the Maven environment. See Codestation (Developers) and Cobertura documentation [<http://cobertura.sourceforge.net/>].
- The source directory where the Cobertura report file is located must be available.
- To have AnthillPro generate a report of all the projects that use Cobertura, an AnthillPro report must first be created.

Configure Cobertura Report Publisher Step

Configure the Cobertura Report Publisher in this section. This step is tasked with retrieving the report generated by Cobertura. You may also choose to publish the data that AnthillPro adds to its database.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.

Before proceeding to Item Four, add Cobertura to your build script, as an Ant task, or as part of the Maven en-

vironment. See Cobertura documentation [<http://cobertura.sourceforge.net/>].

4. **Cobertura Coverage Report.** Select the **Insert After** icon of the step prior to the point where the Cobertura step is to be included (e.g., the Artifact Delivery step). Go to **Coverage > Cobertura**, select the **Cobertura Coverage Report** step, and click **Select**.

- **Name** the step (required).
- **Description.** Provide a short description.
- **Report Name (required).** Give the name for this report (default is same as step name).
- **Report Directory.** Give the directory where the Cobertura HTML report files will be retrieved from.
- **Include Patterns.** Give the file name patterns that describe the files to be retrieved. Each include pattern must be entered on a separate line.

You can also use the following wild cards to tell AnthillPro what to include:

- ****** Indicates include every directory within the base directory.
- ***** Used to include every file. So, if you use `*.zip`, the files matching this pattern will be included.
- ****/*** Tells AnthillPro to retrieve the entire file tree underneath the base directory.
- **Exclude Patterns.** Provide the file name patterns identifying the files that will NOT be retrieved. This field is set in the same way as the Include Patterns field, only you are telling AnthillPro what NOT to include.
- **Store Data.** Check the box to store the coverage.xml results in the AnthillPro database for trending and metrics usage.

If the box is checked, give the base directory where the Cobertura coverage.xml files will be retrieved from if different than the report directory. All files matching the pattern `**/coverage.xml` in the directory will be used.

- **Data Directory.** Give the base directory where the Cobertura coverage.xml files will be retrieved from. Default is the same as Source Directory. All files matching the pattern `**/coverage.xml` in the directory will be used.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

5. Click **Save**.

6. If configuring the **Evaluate Script** step to fail the workflow based on coverage percentage, see Configure Evaluate Script Step (Cobertura). Otherwise, proceed to Add Cobertura Job to Workflow.

Configure Evaluate Script Step (Cobertura)

To have AnthillPro fail a workflow based on the Cobertura report, add an Evaluate Script step to the Build job. The Evaluate Script step, added after the Configure Cobertura Report Publish Step, will read the information in the database and then fail a workflow if the percentage of test coverage is less than the minimum limit. See **Tools > Developer Tools > Scripting API > CoverageHelper**.

Example Coverage Report Script:

```
import com.urbancode.anthill3.domain.coverage.CoverageReport;
import org.apache.log4j.Logger;

static private final Logger log = Logger.getLogger("Stuff");
CoverageReport[] reports = null;
try {
    reports = CoverageHelper.getForCurrentBuildLife();
}
catch (Exception e) {
    log.warn(e);
}

if (reports != null) {
    log.warn("Coverage getForCurrentBuildLife " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}

CoverageReport[] reports = CoverageHelper.getForBuildLife(BuildLifeLookup.getCurrent());
if (reports != null) {
    log.warn("Coverage getForBuildLife " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}

CoverageReport[] reports = CoverageHelper.getForCurrentJobTrace();
if (reports != null) {
    log.warn("Coverage getForCurrentJobTrace " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}

CoverageReport[] reports = CoverageHelper.getForJobTrace(JobTraceLookup.getCurrent());
if (reports != null) {
    log.warn("Coverage getForJobTrace " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}

CoverageReport[] reports = CoverageHelper.getRecentForCurrentWorkflow();
if (reports != null) {
    log.warn("Coverage getRecentForCurrentWorkflow " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
}
```

```
    log.warn("Coverage " + reports[0].getName());
}

CoverageReport[] reports = CoverageHelper.getRecentForWorkflow(WorkflowLookup.getCurrent());
if (reports != null) {
    log.warn("Coverage getRecentForWorkflow " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}
```

To configure the Evaluate Script step:

1. Select the **Insert After** icon of the Cobertura Report Publisher step. Expand the **Miscellaneous folder**, select the **Evaluate Script** step, and click **Select**.
2. **Name** the step (required).
3. **Description**. Provide a short description.
4. **Script**. Give the BeanShell script to evaluate. See Scripting.
5. **Show Additional Options (advanced)**. Select the Show Additional Options link to configure more options.
 - **Is Active**. Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script**. From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures**. Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript**. Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout**. Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
6. Click **Save**.

Add Cobertura Job to Workflow

The Job with the Cobertura step(s) must be executed as part of a workflow (see Configure Cobertura Report Publish Step). This section will assume that an originating workflow has already been configured, and will cover the process of adding the Cobertura build job to the appropriate workflow. Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the Cobertura integration.

1. Go to **Administration**, select the **project**, and select the **build** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.

5. Select the **Build** job created in the Configure Cobertura Report Publish Step section, choose a **job pre-condition** script, and click **Insert Job**.

Run Workflow and View Report (Cobertura)

1. Go to **Dashboard** and select the **workflow** created in the Add Cobertura Job to Workflow section.
2. On the workflow **Main** page, click the **Build** button for the workflow.
3. Once the **workflow** has completed, select the appropriate **Build Life** and click the **Reports** tab.
4. Select the **Cobertura** link to view the Cobertura coverage report for this build.

Coverage Report - com.urbancode.mavenexample

Package /	# Classes	Line Coverage	Branch Coverage	Complexity
com.urbancode.mavenexample	1	60% 9/15 1/2	50% 1/2 1/2	1.2
Classes in this Package /	Line Coverage	Branch Coverage	Complexity	
App	60% 9/15 1/2	50% 1/2 1/2	1.2	

Report generated by [Cobertura](#) 1.9 on 5/1/08 8:51 AM.

com.urbancode.mavenexample

Classes

[App](#) (60%)

5. If the coverage.xml has been made available for report generation, go to the **Reports** page and select the **Run Report** icon to view the coverage report for all AnthillPro projects using Cobertura.

EMMA

Run EMMA and publish coverage reports with the EMMA integration. In addition, AnthillPro users can also fail a workflow based on percentage of code coverage. To use the integration, EMMA must first be added as part of the build, most commonly as an Ant task (see EMMA documentation [<http://emma.sourceforge.net/docs.html>]). Once EMMA has been added, configure the report publisher as a step during job configuration.

EMMA steps:

- **EMMA Coverage Report.** Retrieves the HTML reports generated by EMMA and publishes them on the Build Life. Additionally, AnthillPro can also retrieve the XML reports and store the results in the database for metrics and trending. See Configure EMMA Coverage Report Publisher.
- **Evaluate Script.** Reads the information in the database and then fails a workflow if the percentage of test coverage is less than the minimum limit. See Configure Evaluate Script (EMMA).

Your jobs will vary, but the EMMA integration is added as a job step similar to what is described below. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to an existing job. Once the job is configured, it is then added to a workflow (under the Definition tab). Typically, the EMMA Coverage Report step is included after any unit testing suites, etc.

EMMA Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- EMMA must already be installed and running. See EMMA documentation [<http://emma.sourceforge.net/docs.html>].
- The directory where the EMMA report file is located must be available.
- To have AnthillPro generate a report of all the projects that use EMMA, an AnthillPro report must first be created.

Configure EMMA Coverage Report Publisher

Configure the EMMA Coverage Report publisher step, which is tasked with retrieving the HTML report generated by EMMA. Once EMMA is configured with the builder, the EMMA Coverage Report step is included after the Populate Workspace, Changelog, Stamp, Dependency, Build, and Run Unit Test steps of the typical Build job.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon. If adding EMMA to an existing job, select the appropriate job and proceed to Item Three below.
2. Follow the steps for creating a build job.

Before proceeding to Item Three, add EMMA to your build script, typically as an Ant task. See EMMA documentation [<http://emma.sourceforge.net/docs.html>].

3. **EMMA Coverage Report.** Select the **Insert After** icon of the step prior to the point where the EMMA step is to be included (e.g., the run Unit Tests step). Go to **Coverage > EMMA**, select the **EMMA Coverage Report** step, and click **Select**.

- **Name** the step (required).
- **Description.** Provide a short description.
- **Report Name.** Give the name for this report (default is same as step name).
- **Report Directory.** Give the directory where the EMMA HTML report files will be retrieved from.
- **Include Patterns.** Give the file name patterns that describe the files to be retrieved. Each include pattern must be entered on a separate line.

You can also use the following wild cards to tell AnthillPro what to include:

- ****** Indicates include every directory within the base directory.
- ***** Used to include every file. So, if you use `*.zip`, the files matching this pattern will be included.
- ****/*** Tells AnthillPro to retrieve the entire file tree underneath the base directory.
- **Exclude Patterns.** Provide the file name patterns identifying the files that will NOT be retrieved. This field is set in the same way as the Include Patterns field, only you are telling AnthillPro what NOT to include.
- **Store Data.** Check the box to store the coverage.xml results in the AnthillPro database for metrics usage.
 - **Data Directory.** If the Store Data option is used, give the base directory where the EMMA coverage.xml files will be retrieved from -- if different than the report directory. Default is the same as Source Directory. All files matching the pattern `**/coverage.xml` in the directory will be used.

- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
4. Click **Save**.
 5. If configuring the **Evaluate Script** step to fail the workflow based on coverage percentage, see Configure Evaluate Script Step (EMMA). Otherwise, proceed to Add EMMA Job to Workflow.

Configure Evaluate Script (EMMA)

To have AnthillPro fail a workflow based on the EMMA report, add an Evaluate Script step to the Build job. The Evaluate Script step, added after the EMMA Coverage Report Publish step, will read the information in the database and then fail a workflow if the percentage of test coverage is less than the minimum limit. See **Tools > Developer Tools > Scripting API > CoverageHelper**.

Example Coverage Report Script:

```
import com.urbancode.anthill3.domain.coverage.CoverageReport;
import org.apache.log4j.Logger;

static private final Logger log = Logger.getLogger("Stuff");
CoverageReport[] reports = null;
try {
    reports = CoverageHelper.getForCurrentBuildLife();
}
catch (Exception e) {
    log.warn(e);
}

if (reports != null) {
    log.warn("Coverage getForCurrentBuildLife " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}

CoverageReport[] reports = CoverageHelper.getForBuildLife(BuildLifeLookup.getCurrent());
if (reports != null) {
    log.warn("Coverage getForBuildLife " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}
```



```
CoverageReport[] reports = CoverageHelper.getForCurrentJobTrace();
if (reports != null) {
    log.warn("Coverage getForCurrentJobTrace " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}

CoverageReport[] reports = CoverageHelper.getForJobTrace(JobTraceLookup.getCurrent());
if (reports != null) {
    log.warn("Coverage getForJobTrace " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}

CoverageReport[] reports = CoverageHelper.getRecentForCurrentWorkflow();
if (reports != null) {
    log.warn("Coverage getRecentForCurrentWorkflow " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}

CoverageReport[] reports = CoverageHelper.getRecentForWorkflow(WorkflowLookup.getCurrent());
if (reports != null) {
    log.warn("Coverage getRecentForWorkflow " + reports.length);
    log.warn("Coverage " + reports[0].getLinePercentage());
    log.warn("Coverage " + reports[0].getBranchPercentage());
    log.warn("Coverage " + reports[0].getCoverageType());
    log.warn("Coverage " + reports[0].getName());
}
```

To configure the Evaluate Script step:

1. Select the **Insert After** icon of the EMMA Coverage Report step. Expand the **Miscellaneous folder**, select the **Evaluate Script** step, and click **Select**.
2. **Name** the step (required).
3. **Description**. Provide a short description.
4. **Script**. Give the BeanShell script to evaluate. See Scripting.
5. **Show Additional Options (advanced)**. Select the Show Additional Options link to configure more options.
 - **Is Active**. Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script**. From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures**. Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript**. Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.

- **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

6. Click **Save**.

Add EMMA Job to Workflow

The Job with the EMMA step(s) must be executed as part of a workflow (see Configure EMMA Coverage Report Publisher). This section will assume that an originating workflow has already been configured, and will cover the process of adding the build job that includes EMMA to the appropriate workflow. Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the EMMA integration.

1. Go to **Administration**, select the **project**, and select the **build** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Build** job created in the Configure EMMA Coverage Report Publisher section, choose a **job precondition** script, and click **Insert Job**.

Run Workflow and View Report (EMMA)

1. Go to **Dashboard** and select the **workflow** created in the Add EMMA Job to Workflow section.
2. On the workflow **Main** page, click the **Build** button for the workflow.
3. Once the **workflow** has completed, select the appropriate **Build Life** and click the **Reports** tab.
4. Select the **EMMA** link to view the EMMA coverage report for this build.
5. If the coverage.xml has been made available for report generation (see Store Data in the Configure EMMA Coverage Report Publisher Step section), go to the **Reports** page and select the **Run Report** icon to view the coverage report for all AnthillPro projects using EMMA.

Chapter 67. Source-code Analysis Tools

You can have AnthillPro run your source-code analysis tool and then collect the results. Implemented as job steps, the integrations allow AnthillPro to collect the test output and store it in the AnthillPro data warehouse.

Once set up, the integrations will make the test results available on the Analytics tab, where you can perform trending over time, see which tests are failing, track coverage, and even fail a build based on the captured results.

Checkstyle

Use the Checkstyle integration to integrate source-code analysis into your existing AnthillPro release lifecycle. The integration exposes Checkstyle's findings, tracks changes between builds, references the change log, and makes that information available on the AnthillPro dashboard.

The Checkstyle [<http://checkstyle.sourceforge.net/>] integration is available as an AnthillPro Plugin for version 3.7 and above. To use the integration, it must be added to a job. Once Checkstyle has been added to a job, the Checkstyle report is available on the Dashboard under the Build Life Analytics tab when its workflow run. There, you can view results, track trends, and more.

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbancode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

Checkstyle Prerequisites

- You must have permissions to the System and Administration pages.
- The Checkstyle results must be generated with a formatter type of xml.
- The path to the output file must be available to complete the configuration.

Add Checkstyle to Job

The Checkstyle integration is added to AnthillPro jobs as a step. The Job with the Checkstyle step must be executed as part of a workflow. This section will assume that an originating workflow has already been configured, and will cover the process of adding the Checkstyle build job to the appropriate workflow. Complete workflow configuration is beyond the scope of this entry. The topics covered in detail below are specific to using the Checkstyle integration.

Typically, the Checkstyle step is included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the Build job. However, each job will vary, and it is possible to set up AnthillPro to run Checkstyle as part of a secondary workflow.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.

4. Select the Insert After icon of the step that will run before the Publish Checkstyle Results step.
5. Go to **Source Analytics > Checkstyle**, select **Publish Checkstyle Results**, and click **Select**.
6. Configure step:
 - **Name** the step. This name will be used by the AnthillPro system.
 - **Description**. Give an optional description of this step.
 - **Working Directory Offset (optional)**. Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
 - **Report Name**. Give the name of the report to be published on the AnthillPro dashboard.
 - **Output File**. Give the path to the Checkstyle XML output file from the working directory.
 - **Include Description**. Check the box to include a description on the AnthillPro dashboard for each finding. Please note that if this option is selected, large amounts of space in the database may be required.
 - **Show Environment Variables (optional; advanced)**. Give any optional environment variables in "name=value" format.

Environment variables values may contain references to existing values in the following format: name=\${env/<NAME>} ;value. If the value of the <NAME> variable is "value2" in the current environment, then the above example will be expanded to: name=value2 ;value.

Using this technique, it is possible add an entry to PATH in the following manner: PATH=my/path/entry ;0. Case is significant even on Windows systems.
 - **Show Additional Options (optional; advanced)**. Select the Show Additional Options link to configure more options.
 - **Is Active**. Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script**. From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures**. Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript**. Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout**. Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
7. Click **Save**.
8. See View Checkstyle Results and Trending.

View Checkstyle Results and Trending

Once the job with the Checkstyle step has been configured, run a build. The completed build will publish the report on the Dashboard. To view the results:

- Go to the **Dashboard**, and select the completed **Build Life**.

- On the **Build Life** page, select the **Analytics** tab.

On the Analytics tab, you can explore the number and type of findings, track changes between builds, and find out which defects are new. From the page, it is also possible to reference the change log for more information.

CodeSonar

Use the CodeSonar integration to integrate source-code analysis into your existing AnthillPro release lifecycle. The integration runs CodeSonar, exposes findings, tracks changes between builds, references the change log, and makes that information available on the AnthillPro dashboard.

The CodeSonar integration is available as an AnthillPro Plugin for version 3.7 and above. To use the integration, it must be configured on the AnthillPro System page, and then added to a job. Once the integration has been configured and added to a job, the CodeSonar report is available on the Dashboard under the Build Life Analytics tab. There, you can view results, track trends, and more.

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbancode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

CodeSonar Prerequisites

- You must have permissions to the System and Administration pages.
- CodeSonar must be installed on the AnthillPro Agent machine which will perform the build, scan, etc.
- The **CodeSonar hub host name** and **hub port** must be available to complete the configuration.

Configure CodeSonar

The CodeSonar integration must be configured on the System page before it is added to any jobs.

1. On the **System** page, select **CodeSonar** from the Integration menu.
2. Click **Create New** on the CodeSonar configuration page.
3. Configure integration:
 - **Name.** Give a name for this integration. This name will be used by the AnthillPro system.
 - **Hub Host.** Give the Hub host name. The hub host name is required in order to use the integration.
 - **Hub Port.** Specify the HTTP port on the Hub host given above. The port is required in order to use the integration.
 - **Hub Username (optional).** When AnthillPro runs CodeSonar, it will automatically log into the hub. Depending on how your CodeSonar is configured, a user name (e.g., AnthillPro) may be required.
 - **Hub Password (optional).** Depending on how CodeSonar is configured, a password may be required for AnthillPro to log into the server. If a password is required, give it here. Leave this field blank if a password is not required for AnthillPro to log into the server.

- **Confirm Password.** Retype the password in the field if one is in use.
 - **Command Path (optional).** Give the path to the CodeSonar executables if they are not in the default location (i.e., already on the path). This should only be the path to the directory containing them.
 - **Severity Threshold.** Set the severity threshold that will be used to mark a finding as Critical or not. Since CodeSonar produces a severity number for each finding, any number CodeSonar finding with a number equal to or greater than the threshold set here will be marked as an Error in the results; an finding assigned a number less than the threshold you set here will be marked as Critical. For example, if you use the default value of 10.0, a finding that CodeSonar assigns a severity number of 9 will be marked as Critical in the results; and one assigned a vale of 10.1 will be marked as Error in the results.
4. Click **Save**.
 5. See Add CodeSonar to Job.

Add CodeSonar to Job

The CodeSonar integration is added to AnthillPro jobs as a step. The Job with the CodeSonar step must be executed as part of a workflow. This section will assume that an originating workflow has already been configured, and will cover the process of adding the CodeSonar build job to the appropriate workflow.

Once the job has been created, you will need to ensure that the job runs on the agent machine that has CodeSonar installed on it. This is done by selecting the appropriate agent when configuring the workflow definition. See Complete Build Process Configuration and Agent Management for information on agent selection.

Complete workflow configuration is beyond the scope of this entry. The topics covered in detail below are specific to using the CodeSonar integration.

Typically, the CodeSonar step is included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the Build job. However, each job will vary, and it is possible to set up AnthillPro to run CodeSonar as part of a secondary workflow (if you do, it may be necessary to run a cleanup step prior to running the CodeSonar step).

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.

When configuring the workflow definition, make sure that the job runs on the machine that has both CodeSonar and the AnthillPro Agent installed. See Complete Build Process Configuration and Agent Management for information on agent selection.

4. **Run CodeSonar step.** Select the Insert After icon of the step that will run before the Run CodeSonar step.
5. Go to **Source Analytics > CodeSonar**, select **Run CodeSonar**, and click **Select**.
6. Configure step:
 - **Name** the step. This name will be used by the AnthillPro system.
 - **Description.** Give an optional description of this step.
 - **Working Directory Offset (optional).** Give the working directory to use when executing this command. This

is relative to current working directory. To use the current directory, leave this field blank.

- **Report Name.** Give the name of the CodeSonar report. The report name is used to publish results to the AntHillPro Dashboard. See [View CodeSonar Results and Trending](#).
- **Project Name.** Give the name of the project as set in CodeSonar.
- **No Services.** If CodeSonar *is not* running as a Windows service, check the box to pass the `-no-services` flag so CodeSonar does not run as a service.
- **Command.** Enter the current command(s) used to build the project. If multiple commands are used, each one must begin on a new line.
- **CodeSonar.** Select the correct CodeSonar configuration. AnthillPro supports multiple configurations as part of its CodeSonar integration. For example, it is possible to configure two servers that run different test suites, etc. The names appearing in this drop-down menu were set during the configuration process. If you don't see the correct configuration, see [Configure CodeSonar](#) to add the appropriate configuration before continuing.
- **Show Environment Variables (optional; advanced).** Give any optional environment variables in "name=value" format.

Environment variables values may contain references to existing values in the following format: `name=${env/<NAME>} ;value`. If the value of the `<NAME>` variable is "value2" in the current environment, then the above example will be expanded to: `name=value2 ;value`.

Using this technique, it is possible add an entry to PATH in the following manner: `PATH=my/path/entry ; 0`. Case is significant even on Windows systems.

- **Show Additional Options (optional; advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see [Step Pre-Condition Scripts](#).
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See [Post Processing Scripts](#).
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

7. Click **Save**.

8. See [View CodeSonar Results and Trending](#).

View CodeSonar Results and Trending

Once the job with the CodeSonar step has been configured, run a build. The build will kick off CodeSonar, run the tests, and publish the reports on the Dashboard. To view the results:

- Go to the **Dashboard**, and select the completed **Build Life**.

- On the **Build Life** page, select the **Analytics** tab.

On the Analytics tab, you can explore the number and type of findings, track changes between builds, and find out which defects are new. From the page, it is also possible to reference the change log for more information.

Coverity Prevent

Use the Coverity Prevent integration to integrate source-code analysis into your existing AnthillPro release lifecycle. The integration runs Coverity Prevent, exposes findings, tracks changes between builds, references the change log, and makes that information available on the AnthillPro dashboard.

The Coverity Prevent integration is available as an AnthillPro Plugin for version 3.7 and above. To use the integration, it must be configured on the AnthillPro System page, and then added to a job. Once the integration has been configured and added to a job, the Coverity Prevent report is available on the Dashboard under the Build Life Analytics tab. There, you can view results, track trends, and more.

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbanocode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

Coverity Prevent Prerequisites

- You must have permissions to the System and Administration pages.
- The **Coverity Prevent server remote host name** and **remote port** must be available to complete the configuration.
- Coverity Prevent must be installed and configured **on the same physical machine** as the AnthillPro agent, otherwise the integration will not work.

Configure Coverity Prevent

The Coverity Prevent integration must be configured before it is added to any jobs.

1. On the **System** page, select **Coverity Prevent** from the Integration menu.
2. Click **Create New** on the Coverity Prevent configuration page.
3. Configure integration:
 - **Name.** Give a name for this integration. This name will be used by the AnthillPro system.
 - **Remote Host.** Give the Defect Manager remote host name of the server that results are published to. The server remote host name is required in order to use the integration.
 - **Remote Port.** Specify the HTTP port on the remote Defect Manager host given above. The port is required in order to use the integration.
 - **Username (optional).** When AnthillPro runs Coverity Prevent, it will automatically commit the test results. Depending on how your Defect Manager is configured, a user name (e.g., AnthillPro) may be required.
 - **Password (optional).** Depending on how Coverity Prevent is configured, a password may be required for Ant-

hillPro to log into the server. If a password is required, give it here. Leave this field blank if a password is not required for AnthillPro to log into the server.

- **Confirm Password.** Retype the password in the field if one is in use.
- **Command Path (optional).** Give the path to the Coverity Prevent executables if they are not in the default location (i.e., already on the path). This should only be the path to the directory containing them.

4. Click **Save**.

5. See Add Coverity Prevent to Job.

Add Coverity Prevent to Job

The Coverity Prevent integration is added to AnthillPro jobs as a step. The Job with the Coverity Prevent step must be executed as part of a workflow. This section will assume that an originating workflow has already been configured, and will cover the process of adding the Coverity Prevent job to the appropriate workflow.

Once the job has been created, you will need to ensure that the job runs on the agent machine that has Coverity Prevent installed on it. This is done by selecting the appropriate agent when configuring the workflow definition. See Complete Build Process Configuration and Agent Management for information on agent selection.

Complete workflow configuration is beyond the scope of this entry. The topics covered in detail below are specific to using the Coverity Prevent integration.

Typically, the Coverity Prevent step is included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the Build job. However, each job will vary, and it is possible to set up AnthillPro to run Coverity Prevent as part of a secondary workflow (if you do, it may be necessary to run a cleanup step prior to running the Coverity Prevent step).

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.

When configuring the workflow definition, make sure that the job runs on the machine that has both Coverity Prevent and the AnthillPro Agent installed. See Complete Build Process Configuration and Agent Management for information on agent selection.

4. **Run Coverity Prevent Step.** Select the Insert After icon of the step that will run before the Run Coverity Prevent step.

*Please note that Coverity Prevent must be installed and configured **on the same physical machine** as the AnthillPro agent, otherwise the integration will not work when the job is run. There are a number of ways to ensure that the job runs on the correct agent. For example, you can use a fixed agent filter (see *Configure and Edit Agent Filters*) or an agent filter script to select the appropriate agent.*

5. Go to **Source Analytics > Coverity Prevent**, select **Run Coverity Prevent**, and click **Select**.
6. Configure step:

- **Name** the step. This name will be used by the AnthillPro system.
- **Description.** Give an optional description of this step.

- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **Report Name.** Give the name of the Coverity Prevent report. The report name is used to publish results to the AnthillPro Dashboard. See View Coverity Prevent Results and Trending.
- **Product Name.** Give the name of the product to which AnthillPro will commit the defects.
- **Source Type.** Select the source-code type that will be analyzed. Currently, you can choose C/C++, C#, or Java from the drop-down menu.
- **Command.** Enter the current command(s) used to build the project. If multiple commands are used, each one must begin on a new line.
- **Coverity Prevent.** Select the correct Coverity Prevent configuration. AnthillPro supports multiple configurations as part of its Coverity Prevent integration. For example, it is possible to configure two servers that run different test suites, etc. The names appearing in this drop-down menu were set during the configuration process. If you don't see the correct configuration, see Configure Coverity Prevent to add the appropriate configuration before continuing.
- **Show Environment Variables (optional; advanced).** Give any optional environment variables in "name=value" format.

Environment variables values may contain references to existing values in the following format: `name=${env/<NAME>} ;value`. If the value of the `<NAME>` variable is "value2" in the current environment, then the above example will be expanded to: `name=value2 ;value`.

Using this technique, it is possible add an entry to PATH in the following manner: `PATH=my/path/entry ; 0`. Case is significant even on Windows systems.

- **Show Additional Options (optional; advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

7. Click **Save**.

8. See View Coverity Prevent Results and Trending.

View Coverity Prevent Results and Trending

Once the job with the Coverity Prevent step has been configured, run a build. The build will kick off Coverity Prevent, run the tests, and publish the reports on the Dashboard. To view the results:

- Go to the **Dashboard**, and select the completed **Build Life**.
- On the **Build Life** page, select the **Analytics** tab.

On the Analytics tab, you can explore the number and type of findings, track changes between builds, and find out which defects are new. From the page, it is also possible to reference the change log for more information.

FindBugs

Use the FindBugs [<http://findbugs.sourceforge.net/>] integration to integrate source-code analysis into your existing AnthillPro release lifecycle. The integration exposes FindBugs' findings, tracks changes between builds, references the change log, and makes that information available on the AnthillPro dashboard.

The FindBugs integration is available as an AnthillPro Plugin for version 3.7 and above. To use the integration, it must be added to a job. Once FindBugs has been added to a job, the FindBugs report is available on the Dashboard under the Build Life Analytics tab when its workflow run. There, you can view results, track trends, and more.

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbanocode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

FindBugs Prerequisites

- You must have permissions to the System and Administration pages.
- The FindBugs results must be generated with a formatter type of xml.
- The path to the output file must be available to complete the configuration.

Add FindBugs to Job

The FindBugs integration is added to AnthillPro jobs as a step. The Job with the FindBugs step must be executed as part of a workflow. This section will assume that an originating workflow has already been configured, and will cover the process of adding the FindBugs build job to the appropriate workflow. Complete workflow configuration is beyond the scope of this entry. The topics covered in detail below are specific to using the FindBugs integration.

Typically, the FindBugs step is included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the Build job. However, each job will vary, and it is possible to set up AnthillPro to run FindBugs as part of a secondary workflow.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. Select the Insert After icon of the step that will run before the Publish FindBugs Results step.
5. Go to **Source Analytics > FindBugs**, select **Publish FindBugs Results**, and click **Select**.
6. Configure step:

- **Name** the step. This name will be used by the AnthillPro system.
- **Description.** Give an optional description of this step.
- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **Report Name.** Give the name of the report to be published on the AnthillPro dashboard.
- **Output File.** Give the path to the FindBugs XML output file from the working directory.
- **Include Description.** Check the box to include a description on the AnthillPro dashboard for each finding. Please note that if this option is selected, large amounts of space in the database may be required.
- **Show Environment Variables (optional; advanced).** Give any optional environment variables in "name=value" format.

Environment variables values may contain references to existing values in the following format: name=\${env/<NAME>} ;value. If the value of the <NAME> variable is "value2" in the current environment, then the above example will be expanded to: name=value2 ;value.

Using this technique, it is possible add an entry to PATH in the following manner: PATH=my/path/entry;0. Case is significant even on Windows systems.

- **Show Additional Options (optional; advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

7. Click **Save**.

8. See View Find Bugs Results and Trending.

View FindBugs Results and Trending

Once the job with the FindBugs step has been configured, run a build. When the build is complete, the report will be visible on the Dashboard. To view the results:

- Go to the **Dashboard**, and select the completed **Build Life**.
- On the **Build Life** page, select the **Analytics** tab.

On the Analytics tab, you can explore the number and type of findings, track changes between builds, and find out

which defects are new. From the page, it is also possible to reference the change log for more information.

Fortify 360

Use the Fortify 360 integration to integrate source-code analysis into your existing AnthillPro release lifecycle. The integration exposes Fortify 360's findings, tracks changes between builds, references the change log, and makes that information available on the AnthillPro dashboard.

The Fortify 360 integration is available as an AnthillPro Plugin for version 3.7 and above. To use the integration, it must be configured on the AnthillPro System page, and then added to a job. Once the integration has been configured and added to a job, the Fortify 360 report is available on the Dashboard under the Build Life Analytics tab. There, you can view results, track trends, and more.

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbancode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

Fortify 360 Prerequisites

- Fortify must be installed on the AnthillPro Agent machine which will perform the build, scan, etc.
- You must have permissions to the System and Administration pages.
- The **Fortify 360 Server URL** must be available to complete the configuration. The server URL is only required if you are using the "upload" step.

If you are using Fortify without the 360 server, you will need the Fortify sourceanalyzer installed on the target machine. In addition, you need to ensure that you are not using any "upload" options. Used in this way, AnthillPro will run the scan, parse the results and display the data on its own.

- A user name and password that AnthillPro can use to log into the Fortify 360 server, if required.

Configure Fortify 360

The Fortify 360 integration must be configured before it is added to any jobs.

1. On the **System** page, select **Fortify 360** from the Integration menu.
2. Click **Create New** on the Fortify 360 configuration page.
3. Configure integration:
 - **Name.** Give a name for this integration. This name will be used by the AnthillPro system. For example, the name you give here will be chosen when configuring the Fortify 360 job step.
 - **Server URL.** Give the URL to the Fortify 360 server web interface. This should include protocol and port if needed. The server URL is only required if you are using the "upload" step.

If you are using Fortify without the 360 server, leave this field blank. Note that you will need the Fortify sourceanalyzer installed on the target machine. In addition, you need to ensure that you are not using any "upload" options. Used in this way, AnthillPro will run the scan, parse the results and display the data on its own.

- **Username.** Give the user name to login to the Fortify 360 server. Leave this blank if no user name or password is required.
- **Password.** Give the password associated with the user name above, if your Fortify 360 server requires a password.
 - Confirm password, if one is used.
- **Access Token.** Optionally, you can also use an access token to use instead of user name and password to login to the Fortify 360 server. Note that it is possible to use both a username/password and an access token.
 - Confirm access token, if one is used.
- **Command Path (optional).** Give the path to the sourceanalyzer and fortify client executables if they are not on the path. This should only be the path to the directory containing them. If the executables are not in the default location, you may have to specify the path here.

4. Click **Save**.

5. See Add Fortify 360 to Job.

Add Fortify 360 to Job

The Fortify 360 integration is added to AnthillPro jobs as a step. The Job with the Fortify 360 step must be executed as part of a workflow. The Fortify 360 integration step enables AnthillPro to build and scan the project, and then upload the results to the Fortify server. If a scan or upload is performed, the scan results are also loaded into AnthillPro on the Build Life as a source analytics report.

Once the job has been created, you will need to ensure that the job runs on the agent machine that has Fortify installed on it. This is done by selecting the appropriate agent when configuring the workflow definition. See Complete Build Process Configuration and Agent Management for information on agent selection.

This section will assume that an originating workflow has already been configured, and will cover the process of adding the Fortify 360 build job to the appropriate workflow. Complete workflow configuration is beyond the scope of this entry. The topics covered in detail below are specific to using the Fortify 360 integration.

Typically, the Fortify 360 steps are included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the Build job. However, each job will vary, and it is possible to set up AnthillPro to run Fortify 360 as part of a secondary workflow.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.

When configuring the workflow definition, make sure that the job runs on the machine that has both Fortify and the AnthillPro Agent installed. See Complete Build Process Configuration and Agent Management for information on agent selection.

4. **Fortify Build/Scan/Upload Step.** Select the Insert After icon of the step that will run before the Run Fortify 360 step.

If you are using Fortify without the 360 server, do not use any of the "upload" options.

- Expand the **Source Analytics** folder. Then expand the **Fortify 360** folder.
- Select **Fortify Build/Scan/Upload** and click **Select**.

5. Configure step:

- **Name** the step. This name will be used by the AnthillPro system.
- **Description.** Give an optional description of this step.
- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **Fortify.** Select the correct Fortify 360 configuration. AnthillPro supports multiple configurations as part of its Fortify 360 integration. For example, it is possible to configure two servers that run different test suites, etc. The names appearing in this drop-down menu were set during the configuration process. If you don't see the correct configuration, see Configure Fortify 360 to add the appropriate configuration before continuing.
- **Report Name.** Give the name of the Fortify 360 report. The report name is used to publish results to the AnthillPro Dashboard. See View Fortify 360 Results and Trending.
- **Build ID.** Give the build ID for Fortify 360.
- **Clean.** By default, AnthillPro will perform a cleanup. If you don't want to run a cleanup, de-select this option.
- **Build.** Perform the build. This requires the Build Options (below) to be entered if the default is used.
- **Build Options.** If the build field is checked above, you must give the build options that will be passed to the sourceanalyzer when the build runs. Each argument should be started on a new line.
- **Scan.** By default, a scan is performed. If you don't want AnthillPro to run a scan, de-select this option. If you have AnthillPro run a scan, you will need to give the scan's output file below.
- **Scan File.** The output file of the scan. This is required if performing a scan or uploading. The file name should be in fvdl format. For example: `anthill-fortify-scan.fvdl`.
- **Upload.** By default, AnthillPro will perform the upload to the Fortify server. To perform an upload, you will need give the Project and Version below, as well as give the Scan File above.

If you are using Fortify without the 360 server, leave this field blank. Note that you will need the Fortify sourceanalyzer installed on the target machine. In addition, you need to ensure that you are not using any "upload" options. Used in this way, AnthillPro will run the scan, parse the results and display the data on its own.

- **Project.** Give the name of the project as set in Fortify 360. This is only necessary if the Upload option is selected above.
- **Version.** Give the Project Version name in the Fortify server to upload results to. This is only necessary if the Upload option is selected above.
- **Show Environment Variables (optional; advanced).** Give any optional environment variables in "name=value" format.

Environment variables values may contain references to existing values in the following format: `name=${env/<NAME>} ;value`. If the value of the `<NAME>` variable is "value2" in the current environment, then the above example will be expanded to: `name=value2 ;value`.

Using this technique, it is possible add an entry to PATH in the following manner:

PATH=my/path/entry;0. Case is significant even on Windows systems.

- **Show Additional Options (optional; advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

6. See View Fortify 360 Results and Trending.

View Fortify 360 Results and Trending

Once the job with the Fortify 360 step has been configured, run a build. The build will kick off Fortify 360, run the tests, and publish the reports on the Dashboard. To view the results:

- Go to the **Dashboard**, and select the completed **Build Life**.
- On the **Build Life** page, select the **Analytics** tab.

On the Analytics tab, you can explore the number and type of findings, track changes between builds, and find out which defects are new. From the page, it is also possible to reference the change log for more information.

Klocwork Insight

Use the Klocwork Insight integration to integrate source-code analysis into your existing AnthillPro release life-cycle. The integration exposes Insight's findings, tracks changes between builds, references the change log, and makes that information available on the AnthillPro dashboard.

The Klocwork Insight integration is available as an AnthillPro Plugin for version 3.7 and above. To use the integration, it must be configured on the AnthillPro System page, and then added to a job. Once the integration has been configured and added to a job, the Klocwork Insight report is available on the Dashboard under the Build Life Analytics tab. There, you can view results, track trends, and more.

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbancode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

Klocwork Insight Prerequisites

- You must have permissions to the System and Administration pages.

- The AnthillPro Agent that will run the Klocwork step must be installed on the same machine as Klocwork Insight.
- The **Klocwork Insight Server URL** must be available to complete the configuration.

Configure Klocwork Insight

The Klocwork Insight integration must be configured before it is added to any jobs.

1. On the **System** page, select **Klocwork Insight** from the Integration menu.
2. Click **Create New** on the Klocwork Insight configuration page.
3. Configure integration:
 - **Name.** Give a name for this integration. This name will be used by the AnthillPro system.
 - **Server URL.** Give the URL to the Klocwork Insight server web interface. This should include protocol and port if needed. The server URL is required in order to use the integration.
 - **Command Path (optional).** Give the path to the `kwcheck` and `kwadmin` executables. This should only be the path to the directory containing them. If the executables are not in the default location, you may have to specify the path here.
4. Click **Save**.
5. See Add Klocwork Insight to Job.

Add Klocwork Insight to Job

The Klocwork Insight integration is added to AnthillPro jobs as a step. The Job with the Klocwork Insight step must be executed as part of a workflow. The Klocwork Insight step can be configured to build, analyze and upload the results of any project (C, C++, Java Ant, MS Visual Studio) already configured in Klocwork.

This section will assume that an originating workflow has already been configured, and will cover the process of adding the Klocwork Insight build job to the appropriate workflow.

Once the job has been created, you will need to ensure that the job runs on the machine that has both Klocwork and the AnthillPro Agent (which will run the job) installed on it. This is done by selecting the appropriate agent when configuring the workflow definition. See Complete Build Process Configuration and Agent Management for information on agent selection.

Complete workflow configuration is beyond the scope of this entry. The topics covered in detail below are specific to using the Klocwork Insight integration.

Typically, the Klocwork Insight step is included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the Build job. However, each job will vary, and it is possible to set up AnthillPro to run Klocwork Insight as part of a secondary workflow.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.

When configuring the workflow definition, make sure that the job runs on the machine that has both Klocwork Insight and the AnthillPro Agent installed. See Complete Build Process Configuration and Agent Management for information on agent selection.

4. **Run Klocwork Step.** Select the Insert After icon of the step that will run before the Run Klocwork Insight step.

- Expand the **Source Analytics** folder. Then expand the **Klocwork Insight** folder.
- Select **Run Klocwork** and click **Select**.

5. Configure step:

- **Name** the step. This name will be used by the AnthillPro system.
- **Description.** Give an optional description of this step.
- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **Report Name.** Give the name of the Klocwork Insight report. The report name is used to publish results to the AnthillPro Dashboard. See View Klocwork Insight Results and Trending.
- **Project Name.** Give the name of the project as set in Klocwork Insight.

Note that in order for the integration to work, the project name must start with a letter (i.e., a, b, c, etc.). In addition, only digits (0 through 9), Latin letters (A through Z, a through z) and the underscore character are allowed in a project name. A project name must not be no more than 54 characters.

- **Build.** Perform the build. This requires the Build Options to be entered below.
- **Build Type.** Select the type of project that is being built and analyzed.
- **Build Options.** If the build field is checked above, you must give the build options that will be passed to the sourceanalyzer when the build runs. Each argument should be started on a new line.
- **Build Specification.** Give the path to the Klocwork build specification file, if different than the default kwinject.out location.
- **Run.** By default, the integration will perform the analysis. If this box is checked, you must give the Build Specification above.
- **Include Description (optional).** Check the box to include a description with each Klocwork Insight finding. Note that if this option is selected, it may consume large amounts of space in the database. If you experience problems when including descriptions, change this setting back to the default.
- **Upload.** By default, AnthillPro will perform the upload to the Fortify server. To perform an upload, you will need give the Project and Version below, as well as give the Scan File above.
- **Build Name.** Give the name of the build to load the results as. Typically, the following AnthillPro stamp value is most often used: `${bsh:StampLookup.getLatestStampValue() }`.

Note that only digits (0 through 9), Latin letters (A through Z, a through z) and the underscore character are allowed in a project or build name.

In addition, the AnthillPro Build Life ID can be used for the build name: `${bsh:BuildLifeLookup.getCurrent().getId() }`. As well as the AnthillPro Stamp Value: `${bsh:StampLookup.getLatestStampValue() }`.

- **Klocwork.** Select the correct Klocwork configuration. AnthillPro supports multiple configurations as part of its Klocwork integration. For example, it is possible to configure two servers that run different test suites, etc. The names appearing in this drop-down menu were set during the configuration process. If you don't see the correct configuration, see [Configure Klocwork Insight](#) to add the appropriate configuration before continuing.
- **Show Environment Variables (optional; advanced).** Give any optional environment variables in "name=value" format.

Environment variables values may contain references to existing values in the following format: `name=${env/<NAME>};value`. If the value of the `<NAME>` variable is "value2" in the current environment, then the above example will be expanded to: `name=value2;value`.

Using this technique, it is possible add an entry to `PATH` in the following manner: `PATH=my/path/entry;0`. Case is significant even on Windows systems.

- **Show Additional Options (optional; advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see [Step Pre-Condition Scripts](#).
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See [Post Processing Scripts](#).
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

6. See [View Klocwork Insight Results and Trending](#).

View Klocwork Insight Results and Trending

Once the job with the Klocwork step has been configured, run a build. The build will kick off Klocwork, run the tests, and publish the reports on the Dashboard. To view the results:

- Go to the **Dashboard**, and select the completed **Build Life**.
- On the **Build Life** page, select the **Analytics** tab.

anthillpro by urbancode

Hello admin | [logout](#) | [profile](#) | [announcements](#) | [tools](#) | [help](#)

Dashboard | Current Activity | Search | Reports | Administration | System

Dashboard > Demosthenes Project > Run Tests > BuildLife 46

Main | Reports | Artifacts | Dependencies | Changes | Tests | Coverage | Issues | **Analytics (1)** | Notes

Analytics Reports: [1]

Name: Klocwork
Type: Klocwork
of Metrics: 18

Metrics by Severity

Top Ten Metrics

Metrics

Link	ID	File	Status	Metric (hover for description)	Severity
	1	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagloc_memory.cc, Line 11	Analyze	FNH.MUST	Severe
	10	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagimud.c, Line 54	Analyze	SV.TAINTED.ALLOC.SIZE	Critical
	11	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagimud.c, Line 55	Analyze	SV.TAINTED.ALLOC.SIZE	Critical
	12	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagimud.c, Line 55	Analyze	NPD.FUNC.MUST	Error
	13	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagsmoking_heap.c, Line 18	Analyze	UFM.DEREF.MIGHT	Severe
	14	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagsmoking_heap.c, Line 46	Analyze	UFM.FFM.MIGHT	Severe
	15	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagsoup.c, Line 32	Analyze	ABR	Critical
	16	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagsoup.c, Line 42	Analyze	ABR	Critical
	17	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagzeroes.c, Line 22	Analyze	RNPD.DEREF	Error
	18	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagzeroes.c, Line 64	Analyze	NPD.FUNC.MUST	Error
	2	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagloc_memory.cc, Line 13	Analyze	FMM.MUST	Error
	3	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagloc_memory.cc, Line 15	Analyze	FMM.MUST	Error
	4	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagimemory.c, Line 17	Analyze	MLK.MIGHT	Error
	5	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagimemory.c, Line 17	Analyze	RH.LEAK	Investigate
	6	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagimemory.c, Line 21	Analyze	NPD.FUNC.MIGHT	Error
	7	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagimemory.c, Line 25	Analyze	MLK.MIGHT	Error
	8	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagimud.c, Line 18	Analyze	SV.TAINTED.FMTSTR	Critical
	9	C:\Program Files\Klocwork\Klocwork 8.1 Server\samples\demosthenes\revisions\rev4roach_bagimud.c, Line 34	Analyze	SV.TAINTED.INDEX.ACCESS	Critical

On the Analytics tab, you can explore the number and type of findings, track changes between builds, and find out which defects are new. From the page, it is also possible to reference the change log for more information.

PMD

Use the PMD [<http://pmd.sourceforge.net/>] integration to integrate source-code analysis into your existing AnthillPro release lifecycle. The integration exposes PMD's findings, tracks changes between builds, references the change log, and makes that information available on the AnthillPro dashboard.

The PMD integration is available as an AnthillPro Plugin for version 3.7 and above. To use the integration, it must be added to a job. Once PMD has been added to a job, the PMD report is available on the Dashboard under the Build Life Analytics tab when its workflow run. There, you can view results, track trends, and more.

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbancode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

PMD Prerequisites

- You must have permissions to the System and Administration pages.
- The PMD results must be generated with a formatter type of xml.
- The path to the output file must be available to complete the configuration.

Add PMD to Job

The PMD integration is added to AnthillPro jobs as a step. The Job with the PMD step must be executed as part of a workflow. This section will assume that an originating workflow has already been configured, and will cover the process of adding the PMD build job to the appropriate workflow. Complete workflow configuration is beyond the scope of this entry. The topics covered in detail below are specific to using the PMD integration.

Typically, the PMD step is included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the Build job. However, each job will vary, and it is possible to set up AnthillPro to run PMD as part of a secondary workflow.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. Select the Insert After icon of the step that will run before the Publish PMD Results step.
5. Go to **Source Analytics > PMD**, select **Publish PMD Results**, and click **Select**.
6. Configure step:
 - **Name** the step. This name will be used by the AnthillPro system.
 - **Description**. Give an optional description of this step.
 - **Working Directory Offset (optional)**. Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
 - **Report Name**. Give the name of the report to be published on the AnthillPro dashboard.
 - **Output File**. Give the path to the PMD XML output file from the working directory.
 - **Show Environment Variables (optional; advanced)**. Give any optional environment variables in "name=value" format.

Environment variables values may contain references to existing values in the following format: name=\${env/<NAME>} ; value. If the value of the <NAME> variable is "value2" in the current environment, then the above example will be expanded to: name=value2 ; value.

Using this technique, it is possible add an entry to PATH in the following manner: PATH=my/path/entry ; 0. Case is significant even on Windows systems.
- **Show Additional Options (optional; advanced)**. Select the Show Additional Options link to configure more options.
 - **Is Active**. Select No to temporarily deactivate the step without deleting it; otherwise select Yes.

- **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
- **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
- **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
- **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

7. Click **Save**.

8. See View PMD Results and Trending.

View PMD Results and Trending

Once the job with the PMD step has been configured, run a build. The build will publish the report on the Dashboard when complete. To view the results:

- Go to the **Dashboard**, and select the completed **Build Life**.
- On the **Build Life** page, select the **Analytics** tab.

On the Analytics tab, you can explore the number and type of findings, track changes between builds, and find out which defects are new. From the page, it is also possible to reference the change log for more information.

Sonar

You can use the Sonar integration to have AnthillPro publish your test results (for the most recent build) to the Sonar Dashboard. If you are already using AnthillPro to run the testing tools Sonar supports, you can add the Run Sonar step to your build job, and the test results will be published to the Sonar Dashboard.

The Sonar integration is available as a Plugin for AnthillPro 3.7 and above. To use the integration, it must be configured on the AnthillPro System page, and then added to a job. Once the integration has been configured and added to a job, you can then run it within the context of a workflow.

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbanocode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

Sonar Prerequisites

- You must have permissions to the System and Administration pages, including the ability to configure workflows and jobs.
- The Sonar URL must be available to complete the configuration.
- If you are running Sonar with other than the default database, you will also need to know the:

- name and password Sonar uses to access the database;
- URL to access the database; and
- driver you are using for the Sonar database.

Additionally, if this information is not included in your POM file (or if the project is not configured in Maven), you will need to provide it during configuration.

Configure Sonar

The Sonar integration must be configured on the System page before it is added to any jobs. The information given here will be reused by all of your jobs that include the Run Sonar step. If you have multiple Sonar servers, you will need to create a unique configuration for each one. Then, during job creation, you will select the appropriate configuration.

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbancode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

If you are using Sonar with Derby and the connection is defined in your POM file, you need only provide a name and Sonar URL here. However, if you are using a different database and the connection information is not defined in your POM file, you will need to add some additional configuration.

1. On the **System** page, select **Sonar** from the Integration menu.

If you do not see Sonar in the menu, go to **Server > Plugins** (scroll down on the System page). Make sure that Sonar is active (if the icon in the Active menu is red, click on it). If you do not see the Sonar Plugin, download it [<http://support.urbancode.com/>] from Supportal and then upload the Plugin to your AnthillPro server.

2. Click **Create New** on the Sonar configuration page.

3. Configure the integration:

- **Name.** Give a name for this integration. This name will be used by the AnthillPro system. For example, when configuring the Sonar step, the name given here will be populated in the Sonar drop-down.
- **Sonar URL.** Give the URL AnthillPro will use to connect to Sonar. The URL name is required in order to use the integration. For example: `http://www.mysonar.com:9000`.

If you are running Sonar with a database other than Derby and your database connection is not defined in your Maven POM file (for example, if you are running Sonar in "Light" mode), you will have to give the following (see Add Sonar to Job for more):

- **Database Driver.** Specify the JDBC driver Sonar uses to connect to its database. For example: `com.mysql.jdbc.driver`.
- **Database Username.** Give the name Sonar uses to connect to its database. For example: `Sonar1`, `Sonar2`, etc. This should correspond to the `sonar.jdbc.username` setting.
- **Database Password.** Give the password Sonar uses when connecting to the database. This should correspond to the `sonar.jdbc.password` setting.
- **Confirm Password.** Retype the password in the field.

- **Database URL.** Give the location of the Sonar database. This should correspond to the `sonar.jdbc.url` setting. For example: `jdbc:mysql:www.mysonar.com:9000`.

4. Click **Save**.

5. See Add Sonar to Job.

Add Sonar to Job

The Sonar integration is added to AnthillPro jobs as a step, and must be executed as part of a workflow. In general, the Sonar step is included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, Artifact Delivery steps, as well as any testing-tool steps that may be required. For example, add the Run Sonar step to you build job after your other "publishing" steps.

Actual job configuration will vary; depend on the requirements of your testing tools; and depend on the projects' repository types.

This section assumes an originating workflow has been configured in AnthillPro; that Sonar is already collecting data from your testing tools and publishing the results to the Sonar Dashboard; and that AnthillPro has been configured with Sonar (see Configure Sonar). Complete workflow/job configuration is beyond the scope of this entry. The topics covered in detail below are specific to using the Sonar integration.

Typically, Sonar is not used as part of a Continuous Integration build because the Dashboard only shows data for one build/test run at a time. Running the Sonar job as part of a daily build (i.e., nightly) is recommended unless your specific needs dictate otherwise.

Because Sonar depends on Maven, it may be necessary to configure two different jobs so Sonar will capture the data from all your tests (it may also be necessary to modify how you run your tests). For example, the integration allows you to run Sonar in both regular and "Light" modes, but not during the same job step (see the Sonar documentation for more information):

- The **regular mode** is used when all the projects being tested are in a Maven repository (i.e., already have a POM file).
- The **"Light" mode** is used when your projects do not use a Maven repository. During job configuration, you will have to check "Create POM File" and specify the *Source and Artifact Directories* and a *Project Name* (see Create POM File below).

If you want to run Sonar for both Maven and non-Maven projects, you will have to configure two job steps (possibly two jobs) and then group your tests accordingly.

Once the job has been created, ensure that any agent the Job runs on can establish a connection to Sonar. AnthillPro allows you to do this in many different ways: see Complete Build Process Configuration and Agent Management for information on agent selection.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon. If you are adding the Sonar step to an existing job, select that job and proceed to **Item 4**.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.

When configuring the workflow definition (i.e., adding the job to a workflow), make sure that the job runs on a machine that has access to the Sonar server. See Complete Build Process Configuration and Agent Management for information on agent selection.

4. **Run Sonar step.** Select the Insert After icon of the step that will run before the Run Sonar step. If you are running tests as part of the same job, make sure the Sonar step is add after the tests have run -- usually near the end of the job when AnthillPro is performing publishing tasks.
5. Go to **Source Analytics > Sonar**, select **Run Sonar**, and click **Select**.

6. Configure step:

- **Name** the step. This name will be used by the AnthillPro system.
- **Description.** Give an optional description of this step.
- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **Maven Home.** Give the path to the installation of Maven that will be used. For example: `${MAVEN_2_0_HOME}`. If the Maven home is on the path, leave this field blank.
- **JVM Properties.** If you have any properties that need to be passed to the JVM, give them here. For example, you can add a property to add memory, etc.
- **Sonar.** Select the correct Sonar configuration. AnthillPro supports multiple configurations as part of its Sonar integration. For example, it is possible to configure two servers that run different test suites, etc. The names appearing in this drop-down menu were set during the configuration process. If you don't see the correct configuration, see Configure Sonar to add the appropriate configuration before continuing.
- **Create POM file.** Check the box if your projects are not configured in Maven. This will allow AnthillPro to run Sonar in "Light" mode. If you check this box, AnthillPro will create a POM file at runtime. In order to run Sonar in this manner, you will need to give the following:
 - **Source Directory.** Give the relative path to the directory where the source resides if Create POM File is checked above. Otherwise, leave this field blank. This is required if you are running Sonar in "Light" mode: i.e., the projects you are testing are not in a Maven repository.
 - **Artifact Directory.** Give the relative path to the directory where artifacts are stored if Create POM File is checked above. Otherwise, leave this field blank. This is required if you are running Sonar in "Light" mode: i.e., the projects you are testing are not in a Maven repository.
 - **Project Name.** Give the name to be displayed by Sonar if Create POM File is checked above. Otherwise, leave this field blank. This is required if you are running Sonar in "Light" mode: i.e., the projects you are testing are not in a Maven repository.
- **Show Environment Variables (optional; advanced).** Give any optional environment variables in "name=value" format.

Environment variables values may contain references to existing values in the following format: `name=${env/<NAME>} ;value`. If the value of the `<NAME>` variable is "value2" in the current environment, then the above example will be expanded to: `name=value2 ;value`.

Using this technique, it is possible add an entry to PATH in the following manner: `PATH=my/path/entry ; 0`. Case is significant even on Windows systems.

- **Show Additional Options (optional; advanced).** Select the Show Additional Options link to configure more options.

- **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
7. Click **Save**.
 8. **Run the Sonar workflow.** Go to the AnthillPro Dashboard and select the appropriate workflow. Click **Build**. When the build completes, you can view the results on the Sonar Dashboard.

Chapter 68. Issue Tracking Tools

The issue tracking integrations, implemented as job steps, enable you to associate an issue with a particular build, link to that issue in your issue tracker, and view the status of the issue. The integrations also allow AnthillPro to perform tasks within the issue tracking tool. For example, if your SCM allows you to submit issue tracking information, AnthillPro will extract that information and create a new issue when necessary. (If your SCM does not support this feature, AnthillPro can create an issue based on a regular expression pattern in the commit comment.)

In addition, AnthillPro will collect the output of your issue tracking tool and store it in the AnthillPro data warehouse. This makes it possible for you to publish reports generated by the tool.

For most issue tracking integrations, AnthillPro performs the following tasks:

- **Create a new issue.** You can have AnthillPro automatically create a new issue in your issue tracking system based on conditions you define (e.g., if a failure occurs).
- **Add comments to an issue.** AnthillPro can be configured to parse the current changelog and then add your comment to any issues that it finds.
- **Resolve or update an issue.** When AnthillPro runs a build and finds that an issue status has changed, it can change the status to resolved or another status (depending on the system) based on the issue ID you set.
- **Publish reports.** You can have AnthillPro collect reports from your issue tracking system and then publish that information in the UI.

AccuWork

The AccuWork integration enables AnthillPro to utilize AccuWork's bridge between third-party issue trackers (such as Bugzilla and JIRA) and AccuRev to publish the generated issue report to AnthillPro Build Life. Once the report is published to AnthillPro, it is then made available on the Build Life Issues tab.

AccuWork Step:

- **Publish Issue Report.** Creates a report of AccuWork issues from the current changelog. The step parses the AccuRev change package for any commit comments.

To start using the integration, configure AccuWork integration on the System page. Once that is done, you will typically include the job step near the end of your build job, before the assign status step(s), etc.

AccuWork Prerequisites

- You will need administrative permissions to the System and Administration pages to configure the integration.
- The AccuWork integration must be used in conjunction with your AccuRev projects.
- The agent running the AccuWork job step must be installed on the same machine as the AccuRev client and be able to contact AccuRev.
- The AccuWork issue DB (AccuRev depot) must be available.
- The user name and password AnthillPro will use to connect to AccuWork must already be configured in Accu-

Work.

Configure AccuWork

Before you can start using the integration, you need to tell AnthillPro about AccuWork. This is done on the **System** page, under the Integration menu. Any steps within AnthillPro relying on AccuWork will not work until the integration is configured.

The AccuWork integration requires an issue DB where AccuWork will look for issues as well a user name and password for accessing it.

1. Go to **System > AccuWork** from the Integration menu.
2. On the **AccuWork** page, click **Create New**.
3. Give the following (all fields may contain scripts and/or property lookups):
 - **Issue DB (AccuRev depot)**. Give the AccuWork issue DB that AccuWork will use to find issues.
 - **Issue URL**. Give the AccuWork issue URL. For example, if you provide a URL template such as: `http://accuwork.company.com/browse/${issueId}`, the value `${issueId}` will be replaced in the template with the issue id of the associated issue. This will allow the AnthillPro UI to generate links directly to the AccuWork issue page.
 - **User Name**. Give user name AnthillPro will use to be used to connect to AccuWork. This name must be configured in AccuWork in order for the integration to work.
 - **Password**. The password used to connect to AccuWork. This password must correspond to the AccuWork user name and must be configured in AccuWork.
 - **Password Script (optional)**. To use a script or property lookups for the password, leave the Password field blank and enter it here. See Scripting.
4. Click **Save** then **Done**.
5. See Add AccuWork Step to Job.

Add AccuWork Step to Job

Once the integration is configured, you can add the Publish Issue Report step to your build job. Typically, this will be included near the end of the job. Once the step is added to your job, the next time the build is run and a defect is found, AnthillPro will generate a report and make it available on the Build Life's Issues tab. To configure the job:

1. Go to **Administration** and select the job you want to add the step to.
2. **Add Publish Issue Report step**. Add the step to create a report of AccuWork issues from the build's changelog. The step parses the AccuRev change package to retrieve the commit comments. Insert the step near the end of job, typically before the assign status step(s).

On the Steps page, go to **Issue Tracking > AccuWork**, select the step, and click **Select**.
3. Configure step:
 - **Name**. Give this step a name. For example, Publish AccuWork Issues.

- **Description (optional).** Provide a description of this step.
- **Issue Id Pattern.** Give a regular expression to locate AccuWork-issue ids within the changelog comments. You may add "()"s around the portion of the pattern which identifies the actual Issue ID. The pattern "TST-[0-9]+" would match (returning the same string as issue ids) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1" The pattern "Bug:\[(TST-[0-9]+)\]" would match "Bug:[TST-1]" using bug id TST-1.

In the example, the pattern includes TST, identifies the issues as belonging to AccuWork. Whenever AnthillPro comes across this regular expression, it will automatically generate an issue report for the Build Life.
- **Include Dependencies.** Check the box if you want AnthillPro to include any issues associated with a dependent project. By checking the box, AnthillPro will parse the dependency changelogs when looking for issues.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

4. Click **Save** then **Done**.

5. **Run workflow and view report.** The next time the build runs and an issue is detected, AnthillPro will make the report available on the Build Life Issues tab. From there, you can click on the issue link to go to the AccuWork page.

Bugzilla

Create a new bug, add comments to a bug, and resolve bugs with the Bugzilla 2.22.2 integration. AnthillPro users can also create a report of Bugzilla bugs from the changelog.

In order to use the integration, AnthillPro must first be configured with Bugzilla. The integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab.

Bugzilla Steps:

- **Create Bug.** Create a new bug in Bugzilla. Typically used during a build job.
- **Add Comments.** Add Comments from the current changelog to matching Bugzilla Bugs. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary.
- **Resolve Bug.** Resolve or Close a Bugzilla Issue. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary. Can be used as a post-deployment step added to a deploy workflow to update the state of a Bugzilla bug.

- **Publish Bug Report.** Create a Report of Bugzilla Bugs from the current changelog. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary.

This tutorial will follow a simple project configuration that uses the Create Bug, Add Comment, and Publish Bug Report steps as part of a build workflow. The Resolve Bug step is used as a post-deployment step as part of a deploy workflow. The example in this tutorial uses Subversion, but the basic configuration is similar for any repository type. Your jobs will vary, but the Bugzilla integration is added as a job step similar to what is described below. Though the example goes through the manual creation of a build job, it is possible to use the Job Wizard to create a build job and then manually configure a second job to run as part of the same workflow.

Bugzilla Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- A project must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See Using Life-Cycle Models.

Configure Bugzilla

Let AnthillPro know about Bugzilla. Any steps within AnthillPro relying on Bugzilla will not work until this integration is configured. These fields may all contain scripts and/or property lookups. See Scripting.

1. Go to **System > Bugzilla** from the Integrations menu.
2. On the **Bugzilla Integration** page, click **Edit**.
3. Configure the integration:
 - **Bugzilla Server URL.** Input the Bugzilla server URL.
 - **Bugzilla Issue URL.** You can have AnthillPro automatically generate a link to all of the issues it associates with a Build Life if you give the Issue URL here. Once you give the URL pattern, the issues that appear on the Issues Tab of a Build Life will be linked to the issue in your issue tracker tool for reviewing the issue, adding additional comments, making edits, etc.

Please provide a URL template such as `http://bugs.company.com/browse/${issueId}`. The value `${issueId}` will be replaced in the template with the issue id of the associated issue. This field provides a template which is used throughout AnthillPro to generate links from issues directly to an issue description page within your issue tracker.
 - **User Name.** Give the user name to be used to connect to the Bugzilla server.
 - **Password.** Provide the password to be used to connect to the Bugzilla server.
 - **Password Script.** To use a script or property lookups for the password, leave the Password field blank and enter it here. See Scripting.
 - **Ignore Certificate.** In some instances, the a security certificate error may interfere with the integration (e.g., when using https). If this is the case, check the box.
4. Click **Set** then **Done**.

Configure Bugzilla Jobs

This step will follow a job configuration that adds a comment, publishes a report, creates a bug, and resolves a bug in Bugzilla as part of an AnthillPro workflow. Before configuring jobs, see [Configure Bugzilla](#).

Build Job with Bugzilla Steps

The Add Comments, Publish Bug Report, and Create Bug steps are included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the Build job.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. **Add Comments**. Select the **Insert After** icon of the step prior to the point where the Bugzilla step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > Bugzilla**, select the **Add Comments** step, and click **Select**.

- **Name** the step.
- **Description**. Provide a short description.
- **Bug Id Pattern**. Give a regular expression to locate a Bugzilla bug Id within changelog comments. Add "(" around the portion of the pattern which identifies the actual Issue Id. The pattern "TST-[0-9]+" would match (returning the same string as Issue Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:[(TST-[0-9]+)]" would match "Issue:[TST-1]" using issue Id TST-1.

In the example, the pattern includes TST which identifies the issue as belonging to a Bugzilla project. Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate comment in Bugzilla.

- **Additional Comment**. Give any information to be added to the JIRA comment (in addition to the commit comment).
- **Additional Form Fields**. Provide the enabled Bugzilla preferences which have required attributes. They must be specified in order for the integration to work. For example, target_milestone. See [Bugzilla Documentation \[http://www.bugzilla.org/docs/\]](http://www.bugzilla.org/docs/).
- **Show Additional Options (advanced)**. Select the Show Additional Options link to configure more options.
 - **Is Active**. Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script**. From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see [Step Pre-Condition Scripts](#).
 - **Ignore Failures**. Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript**. Select a script for determining when commands should count as fail or succeed. See [Post Processing Scripts](#).
 - **Timeout**. Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

5. Click **Save**.

6. **Publish Bug Report.** Select the **Insert After** icon of the **Add Comments** step. Go to **Issue Tracking > Bugzilla**, select the **Publish Bug Report** step, and click **Select**.

- **Name** the step.
- **Description.** Provide a short description.
- **Issue Id Pattern.** Give a regular expression to locate a Bugzilla bug Id within changelog comments. Add "(" around the portion of the pattern which identifies the actual Issue Id. The pattern "TST-[0-9]+" would match (returning the same string as Issue Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Bug:\[(TST-[0-9]+)\]" would match "Bug:[TST-1]" using Bug Id TST-1.

In the example, the pattern includes TST, which identifies the issue as belonging to a Bugzilla project. Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate comment in Bugzilla.

- **Include Dependencies.** Check the box to include change logs from dependencies when searching for bug references.
- **Show Additional Options.** See Show Additional Options.

7. Click **Save**.

8. **Create Bug.** Select the **Insert After** icon of the **Publish Bug Report** step. Go to **Issue Tracking > Bugzilla**, select the **Create Bug** step, and click **Select**.

- **Name** the step.
- **Description.** Provide a short description.
- **Product.** Enter the Bugzilla product name.
- **Component.** Give the summary for the bug.
- **Version.** Provide the product version to use for the bug (e.g., AnthillPro 3.4). If left blank this will default to Other.
- **Short Description.** Give a short description of the bug. If left blank, this will default to Anthill3 Bug within Bugzilla.
- **Assigned To.** Provide the assignee for the new bug. If left blank this will default to the Bugzilla User AnthillPro logs in as.
- **Bug File Location.** Enter the environment of this issue.
- **Initial Comment.** Give the initial comment to add upon bug creation.
- **Priority.** Provide the Bugzilla priority.
- **Severity.** Enter the Bugzilla severity.
- **Show Additional Options.** See Show Additional Options.

9. Click **Save**.

Deploy Job with Bugzilla Post-deploy Step

The Resolve Bug step is typically used as part of a post-deployment job. Once the artifacts have been deployed (in

the example there are two artifact sets: Database and Webapp), the Resolve Bug step is run as a separate job in order to ensure that all Bugzilla bugs are resolved.

- While each job is different, every job will typically run an Assign Status step; the Resolve Bug step; a Get Changelog step; a Publish Changelog step; and a Create Stamp Step.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. Select the **Insert After** icon of the step prior to the point where the Bugzilla step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > Bugzilla**, select the **Resolve Bug** step, and click **Select**.
 - **Name** the step.
 - **Description**. Provide a short description.
 - **Bug Key**. Enter the bug key to be resolved in Bugzilla.
 - **Resolution**. Provide the resolution method to use when resolving the bug.
 - **Form Name**. Specify the form names of any Bugzilla preferences that require attributes (e.g., target_milestone). If not specified here, the Bugzilla integration may not work properly.
 - **Show Additional Options**. See Show Additional Options.
5. Click **Save**.

Add Bugzilla Jobs to Workflows

The Jobs created in the Configure Bugzilla Jobs section must be executed as part of a workflow. This section will assume that an originating workflow has already been configured, and will cover the process of adding the Bugzilla job to the appropriate workflows. Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the Bugzilla integration.

Build Workflow with Bugzilla Steps

1. Go to **Administration**, select the **project**, and select the **build** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Build** job created in Configure Bugzilla Jobs, a **job pre-condition** script, and click **Insert Job**.

Deploy Workflow with Bugzilla Post-deploy Step

1. Go to **Administration**, select the **project**, and select the **deploy** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Post Deployment Steps** job created in the Configure Bugzilla Jobs steps, a **job pre-condition** script, and click **Insert Job**.
 - Note the jobs deploy two artifact sets and that the Post Deploy Steps job will change the Bugzilla bug status for both artifact sets.



Run Bugzilla Workflows and View Reports

1. Go to **Dashboard** and select the **workflow** created in the Add Bugzilla Jobs to Workflows section.
2. On the workflow **Main** page, click the **Build** button for the workflow.
 - This procedure is the same for the build and deploy workflows.
3. Once the **workflow** has completed, select the appropriate **Build Life** and click the **Reports** tab.
4. Select a **link** to view a report. To go to the issue that was generated in Bugzilla, follow the **Bug Id** link.

BUGZILLA BUG REPORT

Project: [XPetStore \(SVN\)](#)
 BuildLife: [1180](#)
 Generated On: 04/09/2008 01:02 PM -0400
 Latest Stamp: DEV-730
 Bugzilla Server: <http://192.168.1.126/cgi-bin/bugzilla/>

Bug ID: [22](#)

Name: Test bug
 Product: TestProduct
 Component: TestComponent
 Version: other
 Status: NEW

5. Click the **Issues** tab to view the issues AnthillPro created in Bugzilla.



6. To drill down on each Bugzilla step on the Build Life Summary page, see Trace a Build Life to Source.

ClearQuest

Create a new defect, add comments to a defect, and resolve issues with the IBM Rational ClearQuest integration. AnthillPro users can also create a report of ClearQuest defects from the changelog.

In order to use the integration, AnthillPro must first be configured with ClearQuest. The integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab. See Using the ClearQuest Integration.

ClearQuest Steps:

- **Create Defect.** Create a new defect in a ClearQuest defect tracker. Typically used during a build job.
- **Add Comments.** Add Comments from the current changelog to matching ClearQuest defects. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary.
- **Resolve Issue.** Resolve a ClearQuest defect. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary. Can be used as a post-deployment step added to a deploy workflow to update the state of a ClearQuest defect.

- **Publish Defect Report.** Create a Report of ClearQuest defects from the current changelog. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary.

This tutorial will follow a simple project configuration that uses the Create Defect, Add Comments, and Publish Defect Report steps as part of a build workflow. The Resolve Issue step is used as a post-deployment step as part of a deploy workflow. The example in this tutorial uses Subversion, but the basic configuration is similar for any repository type. Your jobs will vary, but the ClearQuest integration is added as a job step similar to what is described below. Though the example goes through the manual creation of a build job, it is possible to use the Job Wizard to create a build job and then manually configure a second job to run as part of the same workflow.

See also ClearCase.

ClearQuest Prerequisites

- The agent running any ClearQuest jobs must be installed on the same Windows machine as the `cqperl` client.
- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- A project must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See Using Life-Cycle Models.

Configure ClearQuest

Let AnthillPro know about ClearQuest. Any steps within AnthillPro relying on ClearQuest will not work until this integration is configured. These fields may all contain scripts and/or property lookups. See Scripting.

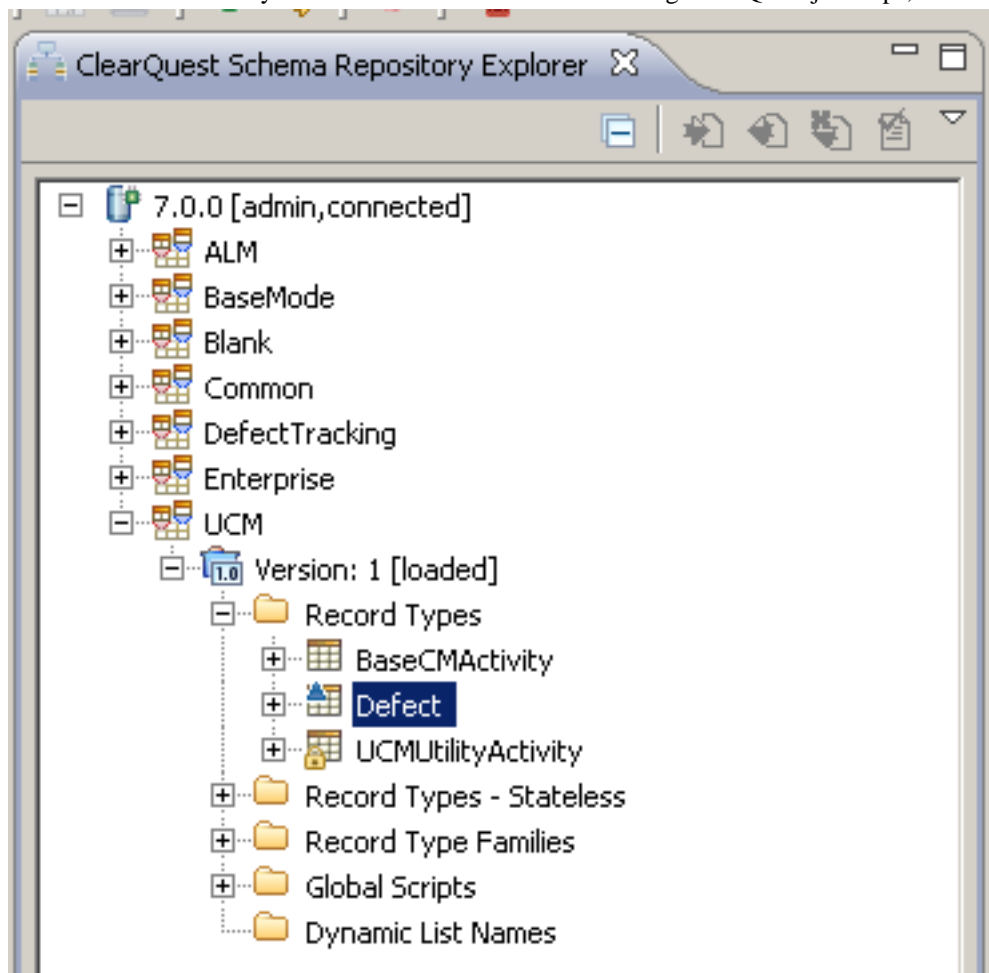
1. Go to **System > ClearQuest** from the Integrations menu.
2. On the **ClearQuest Integration** page, click **Edit**.
3. Configure the integration:
 - **User Name.** Give the user name to be used to connect to the ClearQuest server.
 - **Password.** Provide the password to be used to connect to the ClearQuest server.
 - **Password Script.** To use a script or property lookups for the password, leave the Password field blank and enter it here. See Scripting.
 - **Database Name.** Give the ClearQuest database name.
 - **Database Schema.** Provide the ClearQuest database schema name.
 - **Issue URL.** You can have AnthillPro automatically generate a link to all of the issues it associates with a Build Life if you give the Issue URL here. Once you give the URL pattern, the issues that appear on the Issues Tab of a Build Life will be linked to the issue in your issue tracker tool for reviewing the issue, adding additional comments, making edits, etc.

Please provide a URL template such as `http://bugs.company.com/browse/${issueId}`. The value `${issueId}` will be replaced in the template with the issue id of the associated issue. This field provides a template which is used throughout AnthillPro to generate links from issues directly to an issue description page within your issue tracker.

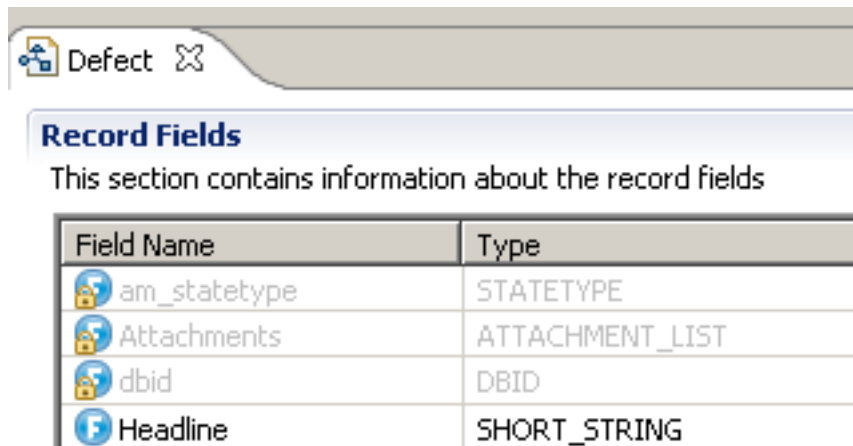
- Select the **Add Mapping** button. Tell AnthillPro about the ClearQuest Schema that will be used. AnthillPro must know the correct **Record Type(s)** and the corresponding fields for the **Name**, **Status**, and **Description** that will be used when AnthillPro executes the ClearQuest job steps. See Using the ClearQuest Integration.

Record Type: *	<input type="text" value="Defect"/>	The Record Type in ClearQuest.
Name Mapping: *	<input type="text" value="Headline"/>	The Name field name of the record type in ClearQuest.
Status Mapping: *	<input type="text" value="State"/>	The Status field name of the record type in ClearQuest.
Description Mapping: *	<input type="text" value="Description"/>	The Description field name of the record type in ClearQuest.

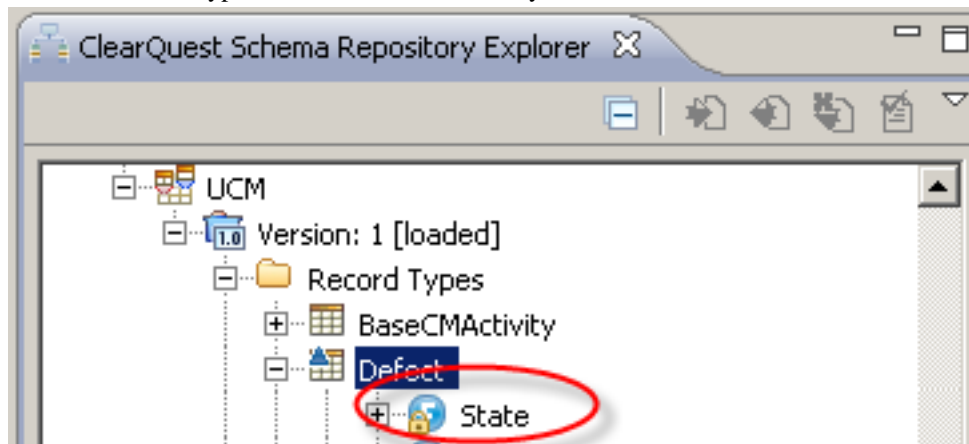
- Record Type.** Enter the ClearQuest Record Type to be used in AnthillPro. For example, if the Record Type "Defect" is to be used by the AnthillPro server when executing ClearQuest job steps, enter it in the field.



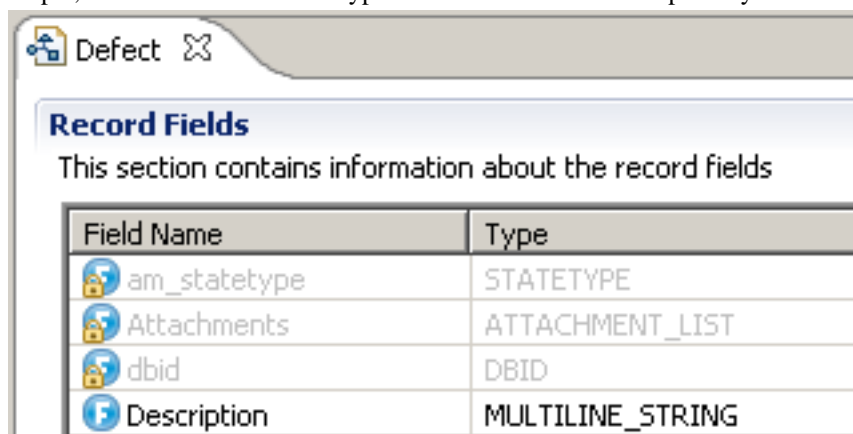
- Name Mapping.** Give the ClearQuest field name that will be used by AnthillPro as the name. For example, if the "Defect" Record Type has a field called "Headline" which you want to use as the name, enter it in the field.



- **Status Mapping.** Give the ClearQuest field name to be used as the status in AnthillPro. For example, if the "Defect" Record Type has a field named "State" you would like to use as the status, enter it in the field.



- **Description Mapping.** Give the ClearQuest field name to be used as the description in AnthillPro. For example, if the "Defect" Record Type has a field named "Description" you would like to use, enter it in the field.



5. Click **Save**.

6. To have AnthillPro use multiple Record Types, repeat Items 4 and 5. For example, add a ClearQuest Record Type "Task" and/or "Story" (along with its corresponding fields) in addition to the "Defect" record type mentioned above.

7. Click **Set** then **Done**.

Using the ClearQuest Integration

This tutorial adds a ClearQuest comment, publishes a ClearQuest report, and creates a defect as part of the build workflow. Complete job configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the ClearQuest integration.

Using the ClearQuest Integration Prerequisites

- The Configure ClearQuest section of this tutorial must be complete.
- You must have AnthillPro administrative privileges to configure the integration. See [Manage Security](#).
- A project with at least one Build Life must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See [Using Life-Cycle Models](#).

Configure ClearQuest Jobs

This step will follow a job configuration that adds a comment, publishes a report, creates a bug, and resolves a bug in ClearQuest as part of AnthillPro workflows.

Build Job with ClearQuest Steps

The Add Comments, Publish Defect Report, and Create Defect steps are included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the Build job.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. **Add Comments**. Select the **Insert After** icon of the step prior to the point where the ClearQuest step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > ClearQuest**, select the **Add Comments** step, and click **Select**.
 - **Name** the step.
 - **Description**. Provide a short description.
 - **Defect Id Pattern**. Give a regular expression to locate a ClearQuest defect Id within changelog comments. Add "()" around the portion of the pattern which identifies the actual Issue Id. The pattern "TST-[0-9]+" would match (returning the same string as Issue Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:[(TST-[0-9]+)]" would match "Issue:[TST-1]" using issue Id TST-1.

In the example, the pattern includes CQDE which identifies the issue as belonging to a ClearQuest project. Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate comment in ClearQuest.
 - **Additional Comment**. Give any information to be added to the JIRA comment (in addition to the commit comment).

- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

5. Click **Save**.

6. **Publish Defect Report.** Select the **Insert After** icon of the **Add Comments** step. Go to **Issue Tracking > ClearQuest**, select the **Publish Defect Report** step, and click **Select**.

- **Name** the step.
- **Description.** Provide a short description.
- **Defect Id Pattern.** Give a regular expression to locate a ClearQuest defect Id within changelog comments. Add "(" around the portion of the pattern which identifies the actual Issue Id. The pattern "TST-[0-9]+" would match (returning the same string as Issue Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Bug:\\[(TST-[0-9]+)\\]" would match "Bug:[TST-1]" using Bug Id TST-1.

In the example, the pattern includes CQDE which identifies the issue as belonging to a ClearQuest project. Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate comment in ClearQuest.

- **Show Additional Options.** See Show Additional Options.

7. Click **Save**.

8. **Create Defect.** Select the **Insert After** icon of the **Publish Defect Report** step. Go to **Issue Tracking > ClearQuest**, select the **Create Defect** step, and click **Select**.

- **Name** the step.
- **Description.** Provide a short description.
- **Headline.** Enter the ClearQuest headline.
- **Severity.** Enter the ClearQuest severity.
- **Project Name.** Provide the ClearQuest product name.
- **Priority.** Provide the ClearQuest priority.
- **Owner.** Provide the owner of the defect.
- **Defect Description.** Enter the description to be sent to ClearQuest.
- **Show Additional Options.** See Show Additional Options.

9. Click **Save**.

Deploy Job with ClearQuest Post-deploy Step

The Resolve Issue step is typically used as part of a post-deployment job. Once the artifacts have been deployed (in the example there are two artifact sets: Database and Webapp), the Resolve Issue step is run as a separate job in order to ensure that all ClearQuest bugs are resolved.

- While each job is different, every job will typically run an Assign Status step; the Resolve issue step; a Get Changelog step; a Publish Changelog step; and a Create Stamp Step.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard).
3. Click **Select**.
4. Follow the steps for creating a build job.
5. Select the **Insert After** icon of the step prior to the point where the ClearQuest step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > ClearQuest**, select the **Resolve Issue** step, and click **Select**.
 - **Name** the step.
 - **Description**. Provide a short description.
 - **Defect Id**. Enter the defect Id to be resolved in ClearQuest.
 - **Resolution**. Provide the resolution that resolved the issue.
 - **Show Additional Options**. See Show Additional Options.
6. Click **Save**.

Add ClearQuest Jobs to Workflows

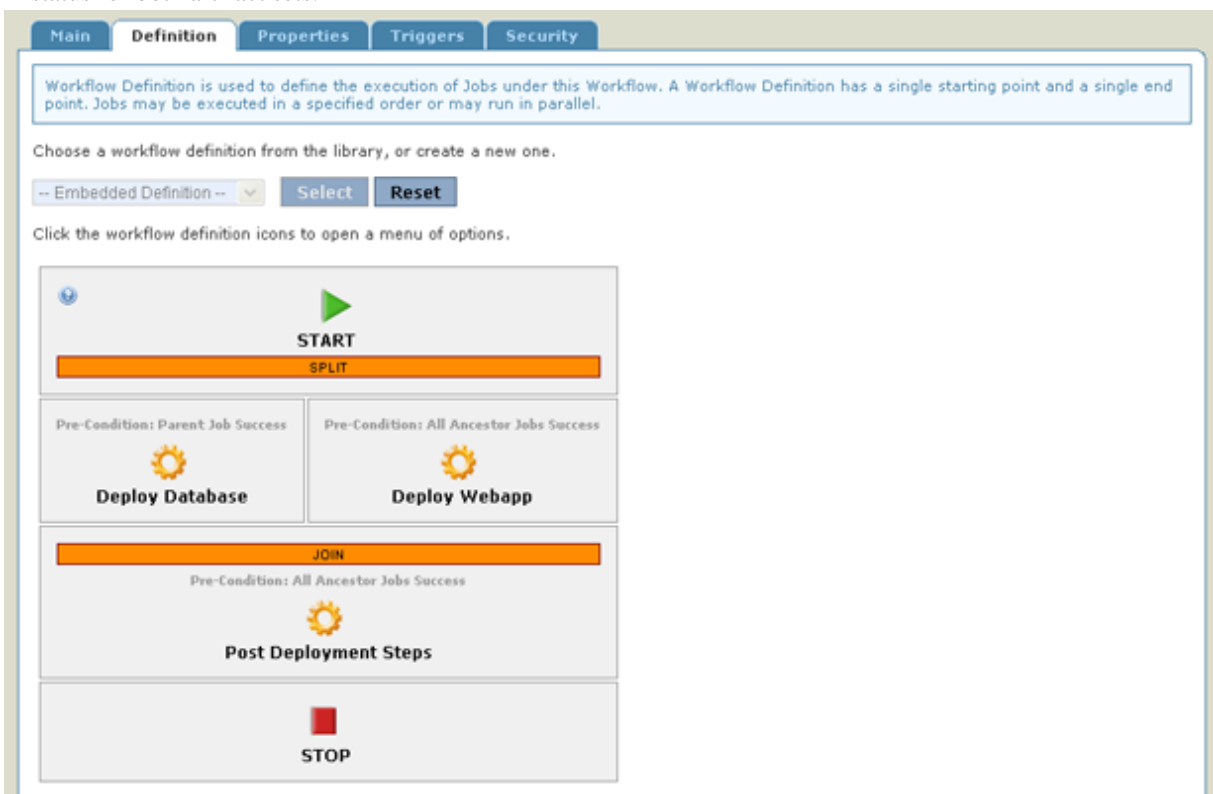
The Jobs created in the Configure Jobs section must be executed as part of a workflow. This section will assume that an originating workflow has already been configured, and will cover the process of adding the ClearQuest job to the appropriate workflows. Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the ClearQuest integration.

Build Workflow with ClearQuest Steps

1. Go to **Administration**, select the **project**, and select the **build** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Build** job created in the Configure Jobs section, a **job pre-condition** script, and click **Insert Job**.

Deploy Workflow with ClearQuest Post-deploy Step

1. Go to **Administration**, select the **project**, and select the **deploy** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Post Deployment Steps** job created in the Configure Jobs section, a **job pre-condition** script, and click **Insert Job**.
 - Note that this jobs deploy two artifact sets and that the Post Deploy Steps job will change the ClearQuest bug status for both artifact sets.



Run ClearQuest Workflows and View Reports

1. Go to **Dashboard** and select the **workflow** created in the Add Jobs to Workflow section.
2. On the workflow **Main** page, click the **Build** button for the workflow.
 - This procedure is the same for the build and deploy workflows.
3. Once the **workflow** has completed, select the appropriate **Build Life** and click the **Reports** tab.
4. Select a **link** to view a report. To go to the issue that was generated in ClearQuest, follow the link.

5. Click the **Issues** tab to view the issues AnthillPro created in ClearQuest.
6. To drill down on each ClearQuest step on the Build Life Summary page, see Trace a Build Life to Source.

CollabNet TeamForge

The CollabNet TeamForge integration allows AnthillPro to automatically create a list of TeamForge artifacts related to source changes; comment on those artifacts; provide a link from an artifact (defect, task, story) to a build; and file new artifacts. The integration also takes advantage of TeamForge's release management features, enabling AnthillPro to: create new TeamForge Releases; upload Codestation files (AP artifacts) to the TeamForge File Release System; and upload a file to the FRS that links back to AnthillPro.

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbanocode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

The integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab. Currently, you can add the following steps to your job:

- **Comment on Tracker Artifacts.** Add comments to the TeamForge Tracker artifacts that are associated with the Build Life. This step is typically used on the originating workflow, as part of your build job.

This requires a Tracker Artifact Report step that *sets the artifacts on the Build Life prior to the comment step*.

- **Create Release.** Create a TeamForge release within a project. This step is typically used on the secondary workflow, as part of your release job.
- **Create Tracker Artifact.** Create a new TeamForge Tracker artifact. This step is typically used on the originating workflow, as part of your build job, if you plan on deploying and running tests as part of the job.
- **Tracker Artifact Report.** Parses the TeamForge Tracker artifact identifiers from the Build Life change comments (source changes) and retrieves information on the artifacts. This step is typically used on the originating workflow, as part of your build job. If you are using a Comment on Tracker Artifacts step, include this step in your job before the comment step.
- **Upload Release Files.** Upload files to a TeamForge release. This step is typically used on the secondary workflow, as part of your release job.

TeamForge Prerequisites

- The TeamForge base URL is needed to complete the integration.
- The user name and password AnthillPro will use must be created in TeamForge prior to using the integration.
- The AnthillPro agent performing the work must be able to communicate with the TeamForge server.

Configure TeamForge

Let AnthillPro know about TeamForge. The information given here will be used by multiple jobs: any steps within AnthillPro relying on TeamForge will not work until this integration is configured.

The integration is written as an AnthillPro Plugin, included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbancode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

If you have multiple installations of TeamForge that AnthillPro must integrate with, configure a separate integration for each. Then, when configuring jobs, select the correct installation.

1. Go to **System** > **TeamForge** from the Integrations menu.
2. On the **TeamForge** page, click **Create New**.
3. Configure the integration:
 - **Name.** Give a unique name for this integration. The name given here will be used throughout the AnthillPro system -- specifically during job creation. If you are configuring integrations with multiple TeamForge installations, ensure that each name is unique.
 - **TeamForge URL.** Enter the base URL to the TeamForge installation base URL: e.g. `http://teamforge.company.com`.
 - **Username.** Give the user name to be used to connect to TeamForge. This must be configured in TeamForge prior to running any jobs using this integration.
 - **Password.** Provide the password associated with the user given above. This must be configured in TeamForge prior to running any jobs using this integration.
 - Confirm password.
4. Click **Save** then **Done**.
5. See Add TeamForge to Project.

Add TeamForge to Project

Once the integration is configured, you can add build and release jobs (that use TeamForge integration) to your project. How you configure the jobs will depend on how your TeamForge platform is set up, as well as on your AnthillPro processes.

If you are unfamiliar with setting up projects and workflows, please see the Getting Started section before continuing. Specifically, the Setting Up a Build and Setting Up a Deployment sections.

There are two basic usage scenarios, one for build workflows and one for release workflows:

- **Steps used with your build job.** In general, you add the following steps to your build job after the actual build steps, near the end of your job, and as part of your originating workflow. In order for the Tracker Artifact Report step to work, it must parse the source changes associated with the build. If AnthillPro finds any appropriate Tracker artifacts, it will then comment on them. The steps may be included in the following order:
 1. **Tracker Artifact Report.** Parses the TeamForge Tracker artifact identifiers from the Build Life change comments (source changes) and retrieves information on the artifacts. This step is typically used on the originating

workflow, as part of your build job. If you are using a Comment on Tracker Artifacts step, include this step in your job before the comment step.

2. **Comment on Tracker Artifacts.** Add comments to the TeamForge Tracker artifacts that are associated with the Build Life. This step is typically used on the originating workflow, as part of your build job, and requires a Tracker Artifact Report step that *sets the artifacts on the Build Life prior to the comment step*.
3. **Create Tracker Artifact (optional).** Create a new TeamForge Tracker artifact. This step is typically used on the originating workflow, as part of your build job, if you plan on deploying and running tests as part of the job.

See Add TeamForge to Build Job (Build Workflow) for more.

- **Steps used with your release job.** In general, you add the following steps to your release job, as part of a secondary workflow. The steps may be included in the following order:
 1. **Create Release.** Create a TeamForge release within a project. This step is typically used on the secondary workflow, as part of your release job.
 2. **Upload Release Files.** Upload files to a TeamForge release. This step is typically used on the secondary workflow, as part of your release job.

See Add TeamForge to Release Job (Secondary Workflow) for more.

Add TeamForge to Build Job (Build Workflow)

Either add the following steps to an existing build job or create a new build job that uses the steps. What follows assumes you are already familiar with basic AnthillPro job configuration (if not, please see Setting Up a Build before continuing). The TeamForge steps, specifically the Tracker Artifact Report step, rely on any Tracker comments found in the source being built, so the steps need to be placed after the actual build steps. Typically, you should include the steps near the end of the job, prior to any assign status steps. To configure a build job:

1. Go to **Administration**, and select the appropriate **project** and either select an existing build job or create a new one. If you are creating a new job, the use the **Create Step** button to add the first job step. Add the following job steps:
2. **Add the Tracker Artifact Report step.** Parses the TeamForge Tracker artifact identifiers from the Build Life change comments (source changes) and retrieves information on the artifacts. This step is typically used on the originating workflow, as part of your build job.

If you are using a Comment on Tracker Artifacts step, include this step in your job before the comment step.

- **Name** the step. This name will be used by the AnthillPro system: e.g., Get TeamForge Tracker Artifacts.
- **Description.** Give an optional description of this step.
- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **Artifact Pattern.** Give the pattern to parse the TeamForge Tracker artifact identifier from the change comments: e.g., `artf[0-9]+`.
- **TeamForge.** Select the correct integration from the drop-down menu (this is the integration set up in the Configure TeamForge section). If you configured multiple integrations on the AnthillPro System page, make sure you select the correct one. Note that it is possible for a single job -- but not a step -- to use different AnthillPro/TeamForge-server configurations.

- **Show Environment Variables (optional; advanced).** Give any optional environment variables in "name=value" format.

Environment variables values may contain references to existing values in the following format: name=\${env/<NAME>} ;value. If the value of the <NAME> variable is "value2" in the current environment, then the above example will be expanded to: name=value2 ;value.

Using this technique, it is possible add an entry to PATH in the following manner: PATH=my/path/entry ; 0. Case is significant even on Windows systems.

- **Show Additional Options (optional; advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see *Step Pre-Condition Scripts* in the User Documentation.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See *Post Processing Scripts* in the User Documentation.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
- Click **Save**.

3. **Add the Comment on Tracker Artifacts step.** Add comments to the TeamForge Tracker artifacts that are associated with the Build Life. This step is typically used on the originating workflow, as part of your build job.

This requires a Tracker Artifact Report step that *sets the artifacts on the Build Life prior to the comment step*.

- **Name** the step. This name will be used by the AnthillPro system.
- **Description.** Give an optional description of this step.
- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **Comment.** Give the template to use for adding a comment to TeamForge Tracker artifacts. For example:

```
AnthillPro found this artifact referenced in a build of ${bsh:
  ProjectLookup.getCurrent().getName()}
- ${bsh:WorkflowLookup.getCurrent().getName()}
Build Life: ${bsh:
  BuildLifeLookup.getCurrent().getId()} - ${bsh:
  UrlHelper.getBuildLifeUrl(BuildLifeLookup.getCurrent())}
Source Revisions: ${bsh:
  import com.urbancode.anthill3.domain.repository.*;
  StringBuilder changeIds = new StringBuilder();
  for (RepositoryChangeSet changeSet : ChangeSetHelper.getChangeSetArray()) {
    changeIds.append(changeSet.getChangeId()).append(" ");
  }
  return changeIds;
}
```

- **TeamForge.** Select the correct integration from the drop-down menu (this is the integration set up in the Configure TeamForge section). If you configured multiple integrations on the AnthillPro System page, make sure you select the correct one. Note that it is possible for a single job -- but not a step -- to use different AnthillPro/TeamForge-server configurations.
 - **Show Environment Variables (optional; advanced).** See Show Environment Variables above.
 - **Show Additional Options (optional; advanced).** See Show Additional Options above.
 - Click **Save**.
4. **Add the Create Tracker Artifact step (optional).** Create a new TeamForge Tracker artifact. This step is typically used on the originating workflow, as part of your build job, if you plan on deploying and running tests as part of the job.
- **Name** the step. This name will be used by the AnthillPro system.
 - **Description.** Give an optional description of this step.
 - **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
 - **Project.** Give the name of the TeamForge project to add the artifact to.
 - **Tracker.** Enter the name of the TeamForge Tracker to add the artifact to: e.g., Defects, Epics, Stories, Tasks, Tests, etc.
 - **Artifact Title.** Provide the title of the artifact.
 - **Artifact Description.** Give a brief description of the artifact.
 - **Artifact Group.** Give the group of the artifact.
 - **Artifact Category.** Enter the category of the artifact.
 - **Artifact Customer.** Give the customer of the artifact.
 - **Artifact Priority.** Select the priority of the artifact.
 - **Artifact Est. Hours.** Give the estimated hours of the artifact.
 - **Artifact Assignee.** Enter the assigned user name of the artifact.
 - **TeamForge.** Select the correct integration from the drop-down menu (this is the integration set up in the Configure TeamForge section). If you configured multiple integrations on the AnthillPro System page, make sure you select the correct one. Note that it is possible for a single job -- but not a step -- to use different AnthillPro/TeamForge-server configurations.
 - **Show Environment Variables (optional; advanced).** See Show Environment Variables above.
 - **Show Additional Options (optional; advanced).** See Show Additional Options above.
 - Click **Save**.
5. **Add job to workflow.** If you are configuring a new build job, you will need to add the job to your build workflow. Complete documentation is outlined in the Setting Up a Build section.

Add TeamForge to Release Job (Secondary Workflow)

Either add the following steps to an existing release job or create a new release job that uses the steps. What follows assumes you are already familiar with basic AnthillPro job configuration. The process for running a TeamForge release is the same as outlined in the Setting Up a Deployment section -- it is run as a secondary process. Typically, the Create Release step should be included prior to the Upload Release File step -- before any of your assign status steps. To configure a release job:

1. Go to **Administration**, and select the appropriate **project** and either select an existing release job or create a new one. When adding steps, you select either the **Insert After** or **Insert Before** icons of existing steps. If you are creating a new job, the use the **Create Step** button to add the first job step. Add the following job steps:

2. **Add the Create Release step.** Create a TeamForge release within a project.

- **Name** the step. This name will be used by the AnthillPro system.
- **Description.** Give an optional description of this step.
- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **Project.** Give the name of the TeamForge project to create in the release: e.g., Brokerage System.
- **Package.** Provide the name of the TeamForge package to create the release in: e.g., Product 1.
- **Release.** Enter the name of the TeamForge release to create: e.g., Release `${bsh:StampLookup.getLatestStampValue() }`. The example, using a simple BeanShell lookup, will append the "Release" with the latest stamp value of the Build Life. If you stamp value is 1.2.3, then the release name passed to TeamForge will be "Release 1.2.3".
- **Status.** Select the status of the TeamForge release to create.
- **TeamForge.** Select the correct integration from the drop-down menu (this is the integration set up in the Configure TeamForge section). If you configured multiple integrations on the AnthillPro System page, make sure you select the correct one. Note that it is possible for a single job -- but not a step -- to use different AnthillPro/TeamForge-server configurations.
- **Show Environment Variables (optional; advanced).** See Show Environment Variables above.
- **Show Additional Options (optional; advanced).** See Show Additional Options above.
- Click **Save**.

3. **Add the Upload Release Files step.** Upload files to a TeamForge release.

- **Name** the step. This name will be used by the AnthillPro system.
- **Description.** Give an optional description of this step.
- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **Project.** Give the name of the TeamForge project to create in the release: e.g., Brokerage System.
- **Package.** Provide the name of the TeamForge package to create the release in: e.g., Product 1.
- **Release.** Enter the name of the TeamForge release to create: e.g., Release `${bsh:StampLookup.getLatestStampValue() }`. The example, using a simple BeanShell lookup,

will append the "Release" with the latest stamp value of the Build Life. If you stamp value is 1.2.3, then the release name passed to TeamForge will be "Release 1.2.3".

- **Base Directory.** Here, you need to give the relative base directory path from the current working directory to use the Includes and Excludes patterns within. If your files are in a "uploadrelease" directory, you would specify `uploadrelease/` here.
- **Includes.** List the files to be retrieved from within the base directory. You can specify the names of files that reside in the base directory: e.g., `AnthillBuildLife.html`. Or, if the files are located in a sub directory, you specify something like `html/AnthillBuildLife.html`. Each include pattern must be entered on a separate line.

You can also use the following wild cards to tell AnthillPro what to include:

- `**` Indicates include every directory within the base directory.
 - `*` Used to include every file. So, if you use `*.html`, the files matching this pattern will be included.
 - `**/*` Tells AnthillPro to retrieve the entire file tree underneath the base directory.
 - **Excludes.** Give the patterns of the files that should be skipped from the include. This field is set in the same way as the Include Artifacts field, only you are telling AnthillPro what NOT to include. If you leave this field blank, AnthillPro will exclude no files.
 - **TeamForge.** Select the correct integration from the drop-down menu (this is the integration set up in the Configure TeamForge section). If you configured multiple integrations on the AnthillPro System page, make sure you select the correct one. Note that it is possible for a single job -- but not a step -- to use different AnthillPro/TeamForge-server configurations.
 - **Show Environment Variables (optional; advanced).** See Show Environment Variables above.
 - **Show Additional Options (optional; advanced).** See Show Additional Options above.
 - Click **Save**.
4. **Add job to workflow.** If you are configuring a new release job, you will need to add the job to your release (secondary) workflow. Complete documentation is outlined in the Setting Up a Deployment section.

JIRA

Create a new issue, add comments to an issue, and update issues with the JIRA integration. AnthillPro users can also create a report of JIRA issues from the changelog. Additionally, you can have AnthillPro publish a link to any JIRA issue page by configuring the Issue URL field. When configured, the data on the Build Life's Issue page becomes a link.

In order to use the JIRA integration, AnthillPro must first be configured to run the appropriate steps within JIRA. The JIRA integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab.

JIRA Steps:

- **Create Issue.** Create a new issue in a JIRA. Typically used during a build job.
- **Add Comments.** Add comments from the current changelog to matching JIRA issues. In order to use this step, a

set working directory, get changelog, and publish changelog step is necessary.

- **Update Issue.** Used to resolve or close a JIRA issue. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary. You can also have AnthillPro pass a comment at the time the issue is updated. Can be used as a post-deployment step added to a deploy workflow to update the state of a JIRA issue.
- **Publish Issue Report.** Create a Report of JIRA issues from the current changelog. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary.

This tutorial will follow a simple project configuration using the JIRA integration steps. The example in this tutorial uses Subversion, but the basic configuration is similar for any repository type. Your jobs will vary, but the JIRA integration is added similar to what is described below. The example goes through the manual creation of a build job; however, it is possible to use the Job Wizard to create a build job and then manually configure a second job to run as part of the same workflow.

Configure JIRA

Let AnthillPro know about JIRA. Any steps within AnthillPro relying on JIRA will not work until this integration is configured. The JIRA integration requires a base URL where the JIRA server is located as well a user name and password for accessing it. These fields may all contain scripts and/or property lookups. See Scripting.

JIRA Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- A project must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See Using Life-Cycle Models.

Configure JIRA

1. Go to **System > JIRA** from the Integrations menu.
2. On the **JIRA Integration** page, click **Edit**.
3. Configure the integration:

- **JIRA Base URL.** Input the JIRA server URL.
- **Issue URL.** You can have AnthillPro automatically generate a link to all of the issues it associates with a Build Life if you give the Issue URL here. Once you give the URL pattern, the issues that appear on the Issues Tab of a Build Life will be linked to the issue in your issue tracker tool for reviewing the issue, adding additional comments, making edits, etc.

Please provide a URL template such as `http://bugs.company.com/browse/${issueId}`. The value `${issueId}` will be replaced in the template with the issue id of the associated issue. This field provides a template which is used throughout AnthillPro to generate links from issues directly to an issue description page within your issue tracker.

- **User Name.** Give the user name to be used to connect to the JIRA server.
- **Password.** Provide the password to be used to connect to the JIRA server.

- **Password Script.** To use a script or property lookups for the password, leave the Password field blank and enter it here. See Scripting.
- **Server Version.** Specify the JIRA server version (e.g., version 3.6.5 to 3.10.x, select 3.6.5).

4. Click **Set** then **Done**.

Using the JIRA Integration

This tutorial will follow a job configuration that adds a JIRA comment, publishes a JIRA report, and creates an issue as part of a build workflow. The update issue step is added as a post-deployment step to a deploy workflow.

Using the JIRA Integration Prerequisites

- The Configure JIRA section of this tutorial must be complete.
- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- A project with at least one Build Life must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See Using Life-Cycle Models.

Configure JIRA Jobs

Two jobs need to be created to use all the JIRA integration steps. (a.) A build job is created with the JIRA comment, publishes a JIRA report, and creates an issue steps. (b.) The update issue step is added as a post-deployment job.

Build Job with JIRA Steps

The Add Comments, Create Issue, and Publish Issue Report steps are included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the typical Build job.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. **Add Comments.** Select the **Insert After** icon of the step prior to the point where the JIRA step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > JIRA**, select the **Add Comments** step, and click **Select**.
 - **Name** the step.
 - **Description.** Provide a short description.
 - **Issue Id Pattern.** Give a regular expression to locate a JIRA defect Id within changelog comments. Add "(" around the portion of the pattern which identifies the actual Issue Id. The pattern "TST-[0-9]+" would match (returning the same string as Issue Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:([(TST-[0-9]+)])" would match "Issue:[TST-1]" using issue Id TST-1.

In the example below, the pattern identifies the issue as belonging to a specific JIRA project. Whenever Ant-

hillPro comes across this regular expression, it will automatically add the appropriate comment in JIRA.

- **Additional Comment.** Give any information to be added to the JIRA comment (in addition to the commit comment).
 - **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
 - Click **Save**.
5. **Create Issue.** Select the **Insert After** icon of the **Add Comments** step. Go to **Issue Tracking > JIRA**, select the **Create Issue** step, and click **Select**. This step will create an issue if a failure occurs.
- **Name** the step.
 - **Description.** Provide a short description.
 - **Project Key.** Enter the JIRA project key.
 - **Assignee.** Provide the assignee for the new issue. An assignee of -1 will use JIRA's automatic assignment.
 - **Summary.** Enter the issue summary.
 - **Environment.** Give the environment for the issue.
 - **Issue Description.** Enter the description.
 - **Show Additional Options.** See Show Additional Options.
 - Click **Save**.
6. **Publish Issue Report.** Select the **Insert After** icon of the **Create Issue** step. Go to **Issue Tracking > JIRA**, select the **Publish Issue Report** step, and click **Select**.
- **Name** the step.
 - **Description.** Provide a short description.
 - **Report Name.** If a new name is not given, the step name will be used.
 - **Issue Id Pattern.** Give a regular expression to locate a JIRA defect Id within changelog comments. Add "()" around the portion of the pattern which identifies the actual Issue Id. The pattern "TST-[0-9]+" would match (returning the same string as Issue Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:([(TST-[0-9]+)])" would match "Issue:[TST-1]" using issue Id TST-1.

- **Include Dependencies.** Check to include change logs from dependencies when searching for issues.
- **Show Additional Options.** See Show Additional Options.

7. Click **Save**.

Deploy Job with JIRA Update Issue Step

The Update Issue step is typically used as part of a post-deployment job. Once the artifacts have been deployed (in the example there are two artifact sets: Database and Webapp), the Update Issue step can be run as a separate job to ensure that all JIRA issues are resolved.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. Select the **Insert After** icon of the step prior to the point where the JIRA step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > JIRA**, select the **Update Issue** step, and click **Select**.
 - **Name** the step.
 - **Description.** Provide a short description.
 - **Action Name.** Give the JIRA action which should be performed on the issues (e.g. Resolve Issue or Close Issue).
 - **Issue Key.** Enter the issue key to be resolved in JIRA.
 - **Issue Id Pattern in Changelog.** Give a regular expression to locate a JIRA defect Id within changelog comments. Add "()" around the portion of the pattern which identifies the actual Issue Id. The pattern "TST-[0-9]+" would match (returning the same string as Issue Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:\\[(TST-[0-9]+)\\]" would match "Issue:[TST-1]" using issue Id TST-1.
 - **Additional Comment.** Give any comments to accompany the updated issue; for example, include the Build Life that the issue was resolved in. To use scripts with this field, see Scripting.
 - **Show Additional Options.** See Show Additional Options.
 - Click **Save**.

Add JIRA Jobs to Workflows

The Jobs created in the Configure Jobs section must be executed as part of a workflow. This section will assume that an originating workflow has already been configured, and will cover the process of adding the JIRA jobs to the appropriate workflows. Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the JIRA integration.

Build Workflow with JIRA Steps

1. Go to **Administration**, select the **project**, and select the **build** workflow.
2. On the workflow **Main** page, select the **Definition** tab.

- From the drop-down menu, choose **Embedded Definition** and click **Select**.
- Left-click the **Start** icon and select **Insert Job After**.
- Select the **Build** job created in the Configure Jobs section, a **job pre-condition** script, and click **Insert Job**.

Deploy Workflow with JIRA Post-deploy Step

- Go to **Administration**, select the **project**, and select the **deploy** workflow.
 - On the workflow **Main** page, select the **Definition** tab.
 - From the drop-down menu, choose **Embedded Definition** and click **Select**.
 - Left-click the **Start** icon and select **Insert Job After**.
 - Select the **Post Deployment Steps** job created in the Configure Jobs section, a **job pre-condition** script, and click **Insert Job**.
- Note that this jobs deploy two artifact sets and that the Post Deploy Steps job will change the JIRA issue status for both artifact sets.



Run Workflows and View Reports (JIRA)

- Go to **Dashboard** and select the **workflow** created in the Create Workflow section.
- On the workflow **Main** page, click the **Build** button for the workflow.

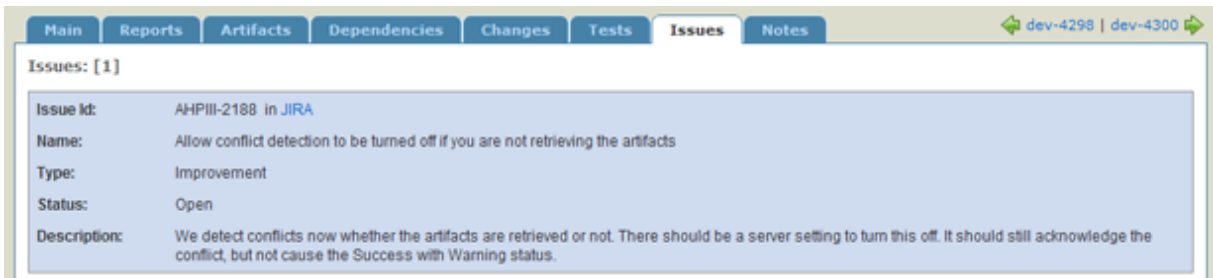
- This procedure is the same for the build and deploy workflows.
3. Once the **workflow** has completed, select the appropriate **Build Life** and click the **Reports** tab.
 4. Select a **link** to view a report. To go to the issue that was generated in JIRA, follow the **Issue Id** link.

JIRA BUG REPORT

Project: [XPetStore \(SVN\)](#)
 BuildLife: [12315](#)
 Generated On: 04/09/2008 10:55 AM -0400
 Latest Stamp: dev-4299
 JIRA Server: <http://bugs.urbancode.com/>

Issue ID: [AHPIII-2188](#)

5. Click the **Issues** tab to view the issues AnthillPro created in JIRA.



6. To drill down on each JIRA step on the Build Life Summary page, see Trace a Build Life to Source.

PVCS Tracker

With the PVCS Tracker integration, AnthillPro users add comments to, resolve, or create a report of PVCS Tracker issues from the changelog.

In order to use the PVCS Tracker integration, AnthillPro must first be configured with an existing PVCS repository. Once the PVCS repository is configured, AnthillPro must then be configured with PVCS Tracker to run the appropriate steps. The PVCS Tracker integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab. See

PVCS Tracker Steps:

- **Add Comments.** Add comments from the current changelog to matching PVCS Tracker issues. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary.
- **Resolve Issue.** Resolve PVCS Tracker issue. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary. Can be used as a post-deployment step added to a deploy workflow to update the state of a PVCS Tracker issue.
- **Publish Issue Report.** Create a Report of PVCS Tracker issues from the current changelog. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary.

This tutorial will follow a simple project configuration using the PVCS Tracker integration steps. The example in this tutorial uses Subversion, but the basic configuration is similar for any repository type. Your jobs will vary, but the PVCS Tracker integration is added as a job step similar to what is described below. The example goes through the manual creation of a build job; however, it is possible to use the Job Wizard to create a build job and then manually configure a second job to run as part of the same workflow.

Configure PVCS Tracker

Let AnthillPro know about PVCS Tracker. Any steps within AnthillPro relying on PVCS Tracker will not work until a PVCS repository and this integration is configured. The PVCS Tracker integration requires the PVCS Tracker server name, a user name, and a password. These fields may all contain scripts and/or property lookups. See [Scripting](#).

PVCS Tracker Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See [Manage Security](#).
- A PVCS repository must be configured with AnthillPro. See [PVCS Source Control](#).
- A project must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See [Using Life-Cycle Models](#).

Configure PVCS Tracker

1. Go to **System > PVCS Tracker** from the Integrations menu.

2. On the **PVCS Tracker Integration** page, click **Edit**.

3. Configure the integration:

- **Server Name.** Input the PVCS Tracker server URL.
- **Issue URL.** You can have AnthillPro automatically generate a link to all of the issues it associates with a Build Life if you give the Issue URL here. Once you give the URL pattern, the issues that appear on the Issues Tab of a Build Life will be linked to the issue in your issue tracker tool for reviewing the issue, adding additional comments, making edits, etc.

Please provide a URL template such as `http://bugs.company.com/browse/${issueId}`. The value `${issueId}` will be replaced in the template with the issue id of the associated issue. This field provides a template which is used throughout AnthillPro to generate links from issues directly to an issue description page within your issue tracker.

- **User Name.** Give the user name to be used to connect to the PVCS Tracker server.
- **Password.** Provide and confirm the password to be used to connect to the PVCS Tracker server.
- **Password Script.** To use a script or property lookups for the password, leave the Password field blank and enter it here. See [Scripting](#).

4. Click **Set** then **Done**.

Using the PVCS Tracker Integration

This tutorial will follow a job configuration that adds a PVCS Tracker comment and publishes a PVCS Tracker report as part of a build workflow. The resolve defect step is added as a post-deployment step to a deploy workflow.

Using the PVCS Tracker Integration Prerequisites

- The Configure PVCS Tracker of this tutorial must be complete.
- You must have AnthillPro administrative privileges to configure the integration. See [Manage Security](#).
- A project with at least one Build Life must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See [Using Life-Cycle Models](#).

Configure PVCS Tracker Jobs

Two jobs need to be created to use all the PVCS Tracker integration steps. (a.) A build job is created with the Add Comments and Publish Issue Report steps. (b.) The Resolve Issue step is added as a post-deployment job.

Build Job with PVCS Tracker Steps

The Add Comments and Publish Issue Report steps are included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the Build job.

1. Go to **Administration** page, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. **Add Comments.** Select the **Insert After** icon of the step prior to the point where the PVCS Tracker step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > PVCS Tracker**, select the **Add Comments** step, and click **Select**.
 - **Name** the step.
 - **Description.** Provide a short description.
 - **Issue Id Pattern.** Give a regular expression to locate a PVCS Tracker defect Id within changelog comments. Add "()" around the portion of the pattern which identifies the actual Issue Id. The pattern "TST-[0-9]+" would match (returning the same string as Issue Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:\\[(TST-[0-9]+)\\]" would match "Issue:[TST-1]" using issue Id TST-1.

In the example below, the pattern identifies the issue as belonging to a specific PVCS Tracker project. Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate comment in PVCS Tracker.

- **Additional Comment.** Use this field to add any additional information to the PVCS Tracker commit comment.
- **Project Name.** Enter the name of the PVCS Tracker project where the issues are to be located.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.

- **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
- **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
- **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
- **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
- **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

5. Click **Save**.

6. **Publisher Issue Report.** Select the **Insert After** icon of the **Add Comments** step. Go to **Issue Tracking > PVCS Tracker**, select the **Publisher Issue Report** step, and click **Select**.

- **Name** the step.
- **Description.** Provide a short description.
- **Report Name.** If a new name is not given, the step name will be used.
- **Defect Id Pattern.** Give a regular expression to locate a PVCS Tracker defect Id within changelog comments. Add "(" around the portion of the pattern which identifies the actual Issue Id. The pattern "TST-[0-9]+" would match (returning the same string as Issue Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:([(TST-[0-9]+)])" would match "Issue:[TST-1]" using issue Id TST-1.
- **Project Name.** Enter the name of the PVCS Tracker project where the issues are to be located.
- **Show Additional Options.** See Show Additional Options.

7. Click **Save**.

Deploy Job with PVCS Tracker Post-deploy Step

The Resolve Issue step is typically used as part of a post-deployment job. Once the artifacts have been deployed (in the example there are two artifact sets: Database and Webapp), the Resolve Issue step is run as a separate job in order to ensure that all PVCS Tracker issues are resolved.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. Select the **Insert After** icon of the step prior to the point where the PVCS Tracker step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > PVCS Tracker**, select the **Resolve Issue** step, and click **Select**.
 - **Name** the step.
 - **Description.** Provide a short description.

- **Defect Id.** Enter the Defect Id to be resolved in PVCS Tracker.
- **State.** Give the PVCS Tracker state the issue is to be moved into.
- **Project Name.** Enter the name of the PVCS Tracker project where the issues are to be located.
- **Show Additional Options.** See Show Additional Options.

5. Click **Save**.

Add PVCS Tracker Jobs to Workflows

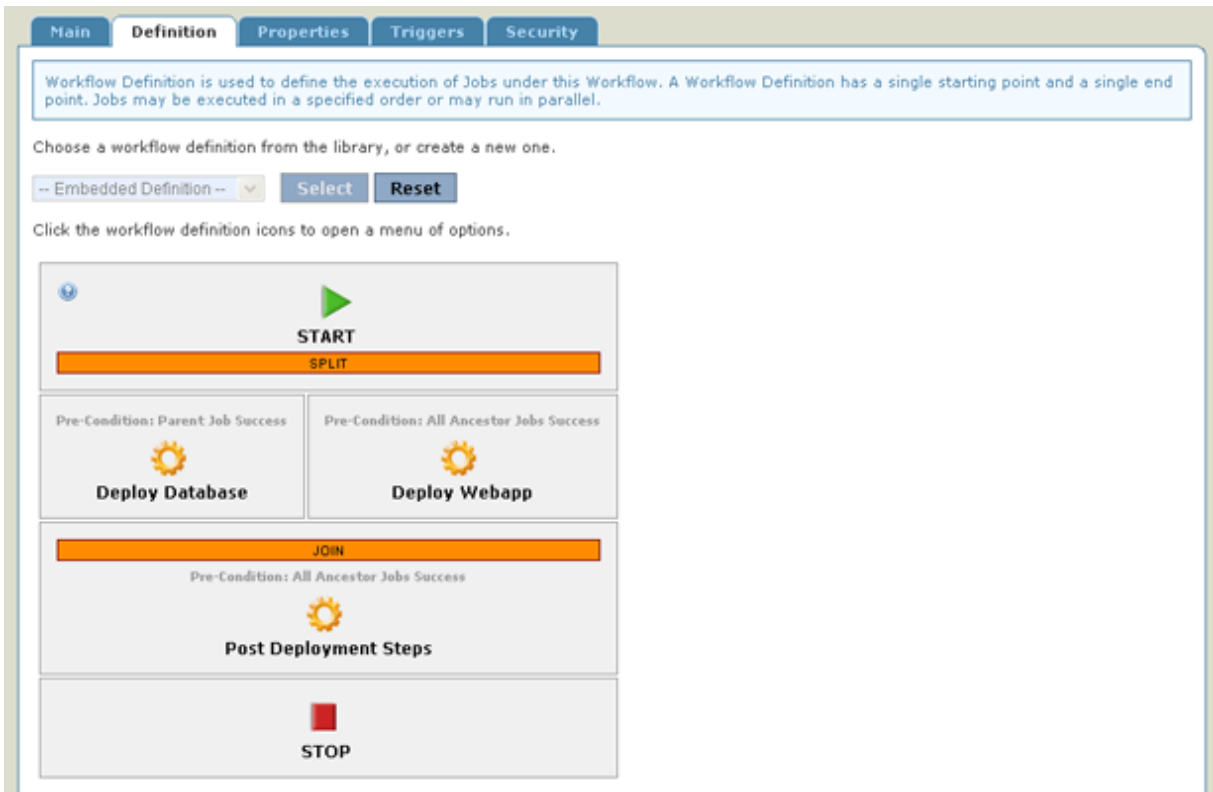
The Jobs created in the Configure Jobs section must be executed as part of a workflow. This section will assume that an originating workflow has already been configured, and will cover the process of adding the PVCS Tracker jobs to the appropriate workflows. Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the PVCS Tracker integration.

Build Workflow with PVCS Tracker Steps

1. Go to **Administration**, select the **project**, and select the **build** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Build** job created in the Configure Jobs section, a **job pre-condition** script, and click **Insert Job**.

Deploy Workflow with PVCS Tracker Post-deploy Step

1. Go to **Administration**, select the **project**, and select the **deploy** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Post Deployment Steps** job created in the Configure Jobs section, a **job pre-condition** script, and click **Insert Job**.
 - Note that this jobs deploy two artifact sets and that the Post Deploy Steps job will change the PVCS Tracker issue status for both artifact sets.



Run Workflows and View Reports (PVCS Tracker)

1. Go to **Dashboard** and select the **workflow** created in the Create Workflow section.
2. On the workflow **Main** page, click the **Build** button for the workflow.
 - This procedure is the same for the build and deploy workflows.
3. Once the **workflow** has completed, select the appropriate **Build Life** and click the **Reports** tab.
4. Select a **link** to view a report. To go to the issue that was generated in PVCS Tracker, follow the **Issue Id** link.
5. Click the **Issues** tab to view the issues AnthillPro created in PVCS Tracker.
6. To drill down on each PVCS Tracker step on the Build Life Summary page, see Trace a Build Life to Source.

Quality Center (Issue Tracking)

The Quality Center integration allows Windows users (through the COM interface) of AnthillPro to add comments, resolve issues, create issues, and publish reports to the AnthillPro UI.

The integration will only work if the agent(s) running the Quality Center steps uses the **32-bit JVM**. If the agent(s) use a 64-bit JVM, the steps will fail. However, the AnthillPro server, which does not run the steps, can run on the 64-bit JVM.

The Quality Center integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab.

Quality Center Issue Tracking Steps:

- **Create Issue.** Creates a new issue in a Quality Center. This step can be included as part of the job (if run on Windows) without any additional steps.
- **Add Comments.** Adds comments from the current changelog to matching Quality Center Issues. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary.
- **Resolve Issue.** Resolves or closes an issue. This step can be used as a workflow property to specify which defect gets resolved with the action the workflow takes. For example, if deploying to QA resolves an issue, use this step and have the Id be a workflow property. Step can be included as part of the job (if run on Windows) without any additional steps. Can be used as a post-deployment step added to a deploy workflow to update the state of an issue.
- **Publish Issue Report.** Creates a report of issues from the current changelog. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary.

The Quality Center integration also allows you to run Quality Center and QTP tests. If you using the QTP integration, you must configure the Quality Center integration on the System page. See Quality Center Testing and QuickTest Pro.

Quality Center Prerequisites (Issue Tracking)

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- The Mercury Quality Center URL must be available.
- AnthillPro must be able to access Quality Center as a user with permissions to run tests.
- An AnthillPro agent must be installed on a Windows machine (XP, 2003, Vista, Windows 7) with the Quality Center API installed. For example, the Quality Center server. This may require configuring a Fixed Agent Filter to ensure the job runs on the appropriate machine. Alternately, the agent with the Quality Center API can be added to the Build Farm. This requires the use of a scripted filter to look for the presence of a variable/property (e.g., qc.agent=true) which must be added to the agent. If this option is chosen, it is advisable to include a filter on every build job that excludes the Quality Center agent. See Configure and Edit Agent Filters.

The integration will only work if the agent(s) running the Quality Center steps uses the **32-bit JVM**. If the agent(s) use a 64-bit JVM, the steps will fail. However, the AnthillPro server, which does not run the steps, can run on the 64-bit JVM.

- The AnthillPro agent that will perform the QualityCenter steps must be installed on the same machine as the **QualityCenter client**. You can download the client from the QualityCenter server, typically located here: `http://<qc_server>:8080/qcbin/ClientSide_index.html`. Follow the instructions on that page to install.

Configure Quality Center (Issue Tracking)

1. Go to **System > Quality Center** under the **Integration** menu.
2. On the **Mercury Quality Center Integration** page, click **Edit**.
3. Configure the integration:
 - Enter the Quality Center server URL.
 - **Issue URL.** You can have AnthillPro automatically generate a link to all of the issues it associates with a Build Life if you give the Issue URL here. Once you give the URL pattern, the issues that appear on the Issues Tab of a Build Life will be linked to the issue in your issue tracker tool for reviewing the issue, adding additional comments, making edits, etc.

For example, provide a URL template such as `http://bugs.company.com/browse/${issueId}`. The value `${issueId}` will be replaced in the template with the issue id of the associated issue. This field provides a template which is used throughout AnthillPro to generate links from issues directly to an issue description page within your issue tracker.
 - **User Name.** Enter the user name to be used to connect to the Quality Center server. Make sure that Quality Center user AnthillPro will use to connect to the server has the appropriate permissions.
 - **Password.** Enter the password to be used to connect to the Quality Center server.
 - **Password Script (optional).** To use a script or property lookups for the password, leave the Password field blank and enter it here. See Scripting.
4. Click **Set** and click **Done**.

Create Quality Center Jobs (Issue Tracking)

Configure the job to automatically add comments, create and resolve issues, and deliver Quality Center bug reports to AnthillPro by setting up the job to *run on the agent with the QC COM API installed*. While each job is different, most jobs typically run a get changelog step; run steps to interact with the changelog and Quality Center; and generate reports and make issue comments.

For example, the Add Comments, Create Issue, and Publish Issue Report steps are included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the typical job.

1. Go to **Administration**, select the appropriate **project**, and configure the job.
2. **Add Comments.** Select the **Insert After** icon of the step prior to the point where the Quality Center step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > Mercury Quality Center**, select the **Add Comments** step, and click **Select**.
 - **Name.** Provide a short name.
 - **Description (optional).** Give a short description.
 - **Issue Id Pattern.** Input a regular expression to locate Quality Center Issue Id within changelog comments. Add "(" around the portion of the pattern identifying the Quality Center Issue Id (which is simply a number). The pattern "TST-[0-9]+" would match (returning the same string as issue Ids) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:([(TST-[0-9]+)])" would match "Issue:[TST-1]" using issue id TST-1.

In the example, the pattern includes `XPSSVNQC` which identifies the issues as belonging to the Xpetstore project (XPS) that uses Subversion (SVN), and is a Quality Center issue (QC). Whenever AnthillPro comes

across this regular expression, it will automatically add the appropriate comment in Quality Center.

- **Additional Comment.** Add any information to be added to the Quality Center comment (in addition to the commit comment).
 - **Domain Name.** Provide the name of the Quality Center Domain where projects are located.
 - **Project Name.** Give the name of the Quality Center Project where issues are located.
 - **Fail Mode.** Select the action you wish AnthillPro to take when an unknown issue ID is encountered.
 - **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
 - Click **Save**.
3. **Publish Issue Report.** Select the **Insert After** icon of the Add Comments step. Go to **Issue Tracking > Mercury Quality Center**, select the **Publish Issue Report** step, and click **Select**.
- **Name.** Provide a short name.
 - **Description (optional).** Give a short description.
 - **Report Name.** Input the name for this report (default is same as step name).
 - **Issue Id Pattern.** Input a regular expression to locate Quality Center Issue Id within changelog comments. Add "(" around the portion of the pattern identifying the Quality Center Issue Id (which is simply a number). The pattern "TST-[0-9]+" would match (returning the same string as issue Ids) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:([(TST-[0-9]+)])" would match "Issue:[TST-1]" using issue id TST-1.
- In the example, the pattern includes *XPSSVNQC* which identifies any the issues as belonging to the Xpetstore project (XPS) that uses Subversion (SVN), and is a Quality Center issue (QC). Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate comment in Quality Center.
- **Include Dependencies.** Check to include change logs from dependencies when searching for Issue References.
 - **Domain Name.** Provide the name of the Quality Center Domain where projects are located. Make sure this corresponds to the settings in the previous step.
 - **Project Name.** Give the name of the Quality Center Project where issues are located. Make sure this corresponds to the settings in the previous step.
 - **Show Additional Options (advanced).** See Show Additional Options.
 - Click **Save**.

4. **Create Issue.** Select the **Insert After** icon of the step prior to the point where the Quality Center step is to be included (e.g., it may be included after an Artifact Delivery step). Go to **Issue Tracking > Mercury Quality Center**, select the **Add Comments** step, and click **Select**.
 - **Name.** Provide a short name.
 - **Description (optional).** Give a short description.
 - **Project Key.** You must include the Quality Center project key for the project being tested.
 - **Assignee.** You must include the Quality Center user this issue must be assigned to. This can be any user with appropriate permissions set in Quality Center.
 - **Summary.** Give the Quality Center summary for this issue.
 - **Issue Description.** Optionally, you can include any information identifying this issue when it is created. For example, you can include a description that identifies the issue was created by AnthillPro, etc.
 - **Detected By.** If you want to use a different name than the default Anthill3, give it here. The name you provide will be displayed in the logs.
 - **Priority.** Optionally, you can have AnthillPro set a priority when the issue is created. Make sure that the priority matches those set in Quality Center.
 - **Severity.** Optionally, you can have AnthillPro set a severity when the issue is created. Make sure that the severity matches those set in Quality Center.
 - **Status.** If you want to use a different status than the default of new, give it here. Make sure that the status matches those set in Quality Center.
 - **Domain Name.** Provide the name of the Quality Center Domain where projects are located. Make sure this corresponds to the settings in the previous step.
 - **Project Name.** Give the name of the Quality Center Project where issues are located. Make sure this corresponds to the settings in the previous step.
 - **Additional Parameters.** Optionally, you can set custom Parameters for the issue. This is entered as a listing of name=value pairs. Each pair must be included on a separate line.
 - **Show Additional Options (advanced).** See Show Additional Options.
 - Click **Save**.

Deploy Job with Quality Center (Issue Tracking) Post-deploy Step

The Resolve Issue step is typically used as part of a post-deployment job that is run after the deploy job, but part of the same workflow. Once the artifacts have been deployed, the Resolve Issue step is run as a separate job in order to ensure that all issues are resolved.

- While each job is different, every job will typically run an Assign Status step; the Resolve Issue step; a Get Changelog step; a Publish Changelog step; and a Create Stamp Step.
1. Go to **Administration**, select the appropriate **project**, and configure the job if not already done.
 2. Select the **Insert After** icon of the step prior to the point where the ClearQuest step is to be included. Go to **Issue**

Tracking > Quality Center, select the **Resolve Issue** step, and click **Select**.

- **Name** the step.
- **Description**. Provide a short description.
- **Issue ID**. Give the issue key to be resolved in Quality Center. You must specify either this or the Issue ID Pattern below.
- **Issue ID Pattern**. If you did not specify the Issue ID, input a regular expression to locate Quality Center Issue ID within changelog comments. Add "(" around the portion of the pattern identifying the Quality Center Issue ID (which is simply a number). The pattern "TST-[0-9]+" would match (returning the same string as issue Ids) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:\\[(TST-[0-9]+)\\]" would match "Issue:[TST-1]" using issue id TST-1.

In the example, the pattern includes *XPSSVNQC* which identifies any the issues as belonging to the Xpetstore project (XPS) that uses Subversion (SVN), and is a Quality Center issue (QC). Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate comment in Quality Center.

- **Status**. If you want to use a different status than the default of new, give it here. Make sure that this matches those set for the Quality Center testing step.
- **Domain Name**. Provide the name of the Quality Center Domain where projects are located. Make sure that this matches those set for the Quality Center testing step.
- **Project Name**. Give the name of the Quality Center Project where issues are located. Make sure that this matches those set for the Quality Center testing step.
- **Show Additional Options**. See Show Additional Options.

3. Click **Save**.

Create Quality Center Workflow (Issue Tracking)

Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the Quality Center integration. The process is the same for both jobs that include the Quality Center steps.

1. Go **Administration** and select the appropriate workflow. You can add the job to either your build workflow or as part of a secondary (i.e., testing) workflow.
2. Go to **workflow Main > Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **job** created in the Create Job section, a **job pre-condition script**, and click **Insert Job**.
6. Select the **Properties** tab and add a workflow property. This will ensure that the job runs on the appropriate agent when using a scripted agent filter. Make sure that the property you set here is also configured on the agent.
 - **Name**. Give the name of the Property (the property <name> will be accessed as `#{property:<name>}`).
 - **Description**. Provide a description for this Property shown when prompting users for value.
 - **Default Value**. Input the value for this property.

- **User May Override.** Check if users are able to specify a new value when running this workflow.
 - **Label.** Provide a label for this Property shown when prompting users for value (leave blank to use the Name as the Label).
 - **Is Required.** Check if a non-empty value for this property is required to run workflow.
 - **Allowed Values.** Give the values users are allowed to select for this property (blank for no restriction of value). Separate each value by entering it on its own line.
7. If you created a post-deploy job with the Resolve step, add it after the original job.
 8. Click **Save**.

Run Build and View Reports (Quality Center Issue Tracking)

1. Go to **Dashboard** and select the appropriate workflow.
2. On the **workflow Main** page, click the **Build** button.
3. Once the workflow is complete, select the **Reports** tab.
4. Select a link to view a report.

Quality Center Bug Report

Project: [XPetStore \(SVN\) QC and QTP](#)
 BuildLife: [1096](#)
 Generated On: 03/14/2008 03:00 PM -0400
 Latest Stamp: DEV-701
 Quality Center Server: <http://rntdqc:8080/qcbin>

Issue ID: 52

Name: Change the background of my application to red
 Description: Change the background color to red as a demo.
 Status: New
 Assigned To: etm
 Detected By: etm
 Priority:

5. To drill down on each Quality Center step on the Build Life Summary page, see Trace a Build Life to Source.

Quality Center Function Calls

Following is a list of the function calls used in the Quality Center integration:

TDApi-Ole80.TDConnection	TestSet.StartExecution	TestSet.TSTestFactory	Step.Field
--------------------------	------------------------	-----------------------	------------

TDConnection.BugFactory	TSScheduler.RunAllLocally	TSTestFactory.NewList	BugFactory.Filter
TDConnection.InitConnectionEx	TSScheduler.TdHostName	TSTest.Name	BugFactory.Item
TDConnection.Login	TSScheduler.Run	TSTest.Type	BugFactory.AddItem
TDConnection.Connect	TSScheduler.ExecutionStatus	TSTest.Status	Filter.Filter
TDConnection.Disconnect	ExecutionStatus.RefreshExecStatusInfo	TSTest.HostName	Filter.NewList
TDConnection.Logout	ExecutionStatus.Finished	TSTest.TestId	Bug.ID
TDConnection.ReleaseConnection	ExecutionStatus.EventsList	TSTest.TestName	Bug.Summary
TDConnection.TestSetTreeManager	ExecEventInfo.EventType	TSTest.LastRun	Bug.Field
TestSetTreeManager.NodeByPath	ExecEventInfo.EventDate	Run.Name	Bug.AssignedTo
TestSetFolder.FindTestSets	ExecEventInfo.EventTime	Run.Status	Bug.DetectedBy
TestSet.Id	TestExecStatus.TestId	Run.Field	Bug.Priority
TestSet.Item	TestExecStatus.TestInstance	Run.StepFactory	Bug.Project
TestSet.Name	TestExecStatus.TsTestId	StepFactory.NewList	Bug.Status
TestSet.TestSetFolder	TestExecStatus.Message	Step.Name	Bug.Post
TestSet.Status	TestExecStatus.Status	Step.Status	

Quality Center Plugin (Issue Tracking)

The Quality Center integration allows Windows users (through the COM interface) of AnthillPro to add comments, resolve issues, create issues, and publish reports to the AnthillPro UI.

The integration is written as an AnthillPro Plugin, and expands upon the existing Quality Center (Issue Tracking) integration. The integration is included in the normal distribution. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbanocode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

The Quality Center integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job.

The integration will only work if the agent(s) running the Quality Center steps uses the **32-bit JVM**. If the agent(s) use a 64-bit JVM, the steps will fail. However, the AnthillPro server, which does not run the steps, can run on the 64-bit JVM.

Once the job is configured, it is then added to the workflow under the Definition tab.

Quality Center Issue Tracking Steps:

- **Create Quality Center Issue.** Creates a new issue in a Quality Center. This step can be included as part of the

job (if run on Windows) without any additional steps.

- **Add Comments to Quality Center Issue.** Adds comments from the current changelog to matching Quality Center Issues. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary.
- **Update Quality Center Issue.** Resolves or closes an issue. This step can be used as a workflow property to specify which defect gets resolved with the action the workflow takes. For example, if deploying to QA resolves an issue, use this step and have the Id be a workflow property. Step can be included as part of the job (if run on Windows) without any additional steps. Can be used as a post-deployment step added to a deploy workflow to update the state of an issue.
- **Publish Quality Center Issue Report.** Creates a report of issues from the current changelog. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary.

The Quality Center integration also allows you to run Quality Center and QTP tests. If you using the QTP integration, you must configure the Quality Center integration on the System page. See Quality Center Plugin (Testing) and QuickTest Pro.

Quality Center Plugin Prerequisites (Issue Tracking)

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- The Mercury Quality Center URL must be available.
- AnthillPro must be able to access Quality Center as a user with permissions to run tests.
- An AnthillPro agent must be installed on a Windows machine (XP, 2003, Vista, Windows 7) with the Quality Center API installed. For example, the Quality Center server. This may require configuring a Fixed Agent Filter to ensure the job runs on the appropriate machine. Alternately, the agent with the Quality Center API can be added to the Build Farm. This requires the use of a scripted filter to look for the presence of a variable/property (e.g., qc.agent=true) which must be added to the agent. If this option is chosen, it is advisable to include a filter on every build job that excludes the Quality Center agent. See Configure and Edit Agent Filters.

The integration will only work if the agent(s) running the Quality Center steps uses the **32-bit JVM**. If the agent(s) use a 64-bit JVM, the steps will fail. However, the AnthillPro server, which does not run the steps, can run on the 64-bit JVM.

- The AnthillPro agent that will perform the QualityCenter steps must be installed on the same machine as the **QualityCenter client**. You can download the client from the QualityCenter server, typically located here: `http://<qc_server>:8080/qcbin/ClientSide_index.html`. Follow the instructions on that page to install.

Configure Quality Center Plugin (Issue Tracking)

The information given here will be used by your AnthillPro projects. If you are using both the Quality Center (Testing) and Quality Center (Issue Tracking) Plugins, you need only configure the integration once (assuming you have only one Quality Center server) -- both integrations use the same System configuration.

If you are configuring integrations with multiple Quality Center servers, create a new integration for each one.

1. Go to **System > Quality Center Plugin** under the **Integration** menu.
2. On the **Quality Center Plugin** page, click **Create New**.
3. Configure the integration:
 - **Name.** Give a unique name for this integration. The name given here will be used throughout the AnthillPro system -- specifically during job creation. If you are configuring integrations with multiple Quality Center servers, ensure that each name is unique.
 - **Server URL.** Enter the base URL to the TeamForge installation base URL: e.g. `http://qualitycenter.company.com`.
 - **User Name.** Enter the user name to be used to connect to the Quality Center server. Make sure that Quality Center user AnthillPro will use to connect to the server has the appropriate permissions.
 - **Password.** Enter the password to be used to connect to the Quality Center server. If using a password script below, leave this field blank.
 - Confirm password.
 - **Password Script.** To use a script or property lookups for the password, leave the Password field blank and enter it here. See Scripting.
4. Click **Set** and click **Done**.

Create Quality Center Plugin Jobs (Issue Tracking)

Configure the job to automatically add comments, create and resolve issues, and deliver Quality Center bug reports to AnthillPro by setting up the job to *run on the agent with the QC COM API installed*. While each job is different, most jobs typically run a get changelog step; run steps to interact with the changelog and Quality Center; and generate reports and make issue comments.

For example, the Add Comments, Create Issue, and Publish Issue Report steps are included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the typical job.

1. Go to **Administration**, select the appropriate **project**, and configure the job.
2. **Add Comments to Quality Center Issue.** Select the **Insert After** icon of the step prior to the point where the Quality Center step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > Quality Center Plugin**, select the **Add Comments to Quality Center Issue** step, and click **Select**.
 - **Name.** Provide a short name.
 - **Description (optional).** Give a short description.
 - **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
 - **QC Server.** Select the correct integration from the drop-down menu (this is the integration set up in the Configure Quality Center Plugin (Issue Tracking) section). If you configured multiple integrations on the AnthillPro System page, make sure you select the correct one. Note that it is possible for a single job -- but not a step -- to use different AnthillPro/Quality-Center-server configurations.
 - **Domain Name.** Provide the name of the Quality Center Domain where projects are located.
 - **Project Name.** Give the name of the Quality Center Project where issues are located.

- **Issue Id Pattern.** Input a regular expression to locate Quality Center Issue Id within changelog comments. Add "(" around the portion of the pattern identifying the Quality Center Issue Id (which is simply a number).

The default value, `QC-([0-9]+)` is a regex that parses the change log and applies the attributes of the step to matching issues. For example, if QC-52 was located in the comment of a change, this step would act on issue 52 in the Quality Center database.

- **Additional Comment.** Add any information to be added to the Quality Center comment (in addition to the commit comment).
- **Fail Mode.** Select the action you wish AnthillPro to take when an unknown issue ID is encountered.
- **Show Environment Variables (optional; advanced).** Give any optional environment variables in "name=value" format.

Environment variables values may contain references to existing values in the following format: `name=${env/<NAME>} ; value`. If the value of the `<NAME>` variable is "value2" in the current environment, then the above example will be expanded to: `name=value2 ; value`.

Using this technique, it is possible add an entry to `PATH` in the following manner: `PATH=my/path/entry ; 0`. Case is significant even on Windows systems.

- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
- Click **Save**.

3. **Publish Quality Center Issue Report.** Select the **Insert After** icon of the Add Comments step. Go to **Issue Tracking > Quality Center Plugin**, select the **Publish Quality Center Issue Report** step, and click **Select**.

- **Name.** Provide a short name.
- **Description (optional).** Give a short description.
- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **QC Server.** Select the correct integration from the drop-down menu (this is the integration set up in the Configure Quality Center Plugin (Issue Tracking) section). If you configured multiple integrations on the AnthillPro System page, make sure you select the correct one. Note that it is possible for a single job -- but not a step -- to use different AnthillPro/Quality-Center-server configurations.
- **Domain Name.** Provide the name of the Quality Center Domain where projects are located. Make sure this corresponds to the settings in the previous step.

- **Project Name.** Give the name of the Quality Center Project where issues are located. Make sure this corresponds to the settings in the previous step.
- **Issue Id Pattern.** Input a regular expression to locate Quality Center Issue Id within changelog comments. Add "()" around the portion of the pattern identifying the Quality Center Issue Id (which is simply a number).

The default value, `QC-([0-9]+)` is a regex that parses the change log and applies the attributes of the step to matching issues. For example, if QC-52 was located in the comment of a change, this step would act on issue 52 in the Quality Center database.

- **Fail Mode.** Select the action you wish AnthillPro to take when an unknown issue ID is encountered.
- **Show Environment Variables (optional; advanced).** See Show Environment Variables above.
- **Show Additional Options (advanced).** See Show Additional Options.
- Click **Save**.

4. **Create Quality Center Issue.** Select the **Insert After** icon of the step prior to the point where the Quality Center step is to be included (e.g., it may be included after an Artifact Delivery step). Go to **Issue Tracking > Quality Center Plugin**, select the step, and click **Select**.

- **Name.** Provide a short name.
- **Description (optional).** Give a short description.
- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **QC Server.** Select the correct integration from the drop-down menu (this is the integration set up in the Configure Quality Center Plugin (Issue Tracking) section). If you configured multiple integrations on the AnthillPro System page, make sure you select the correct one. Note that it is possible for a single job -- but not a step -- to use different AnthillPro/Quality-Center-server configurations.
- **Domain Name.** Provide the name of the Quality Center Domain where projects are located. Make sure this corresponds to the settings in the previous step.
- **Project Name.** Give the name of the Quality Center Project where issues are located. Make sure this corresponds to the settings in the previous step.
- **Project Key.** You must include the Quality Center project key for the project being tested.
- **Assignee.** You must include the Quality Center user this issue must be assigned to. This can be any user with appropriate permissions set in Quality Center.
- **Summary.** Give the Quality Center summary for this issue.
- **Detected By.** If you want to use a different name than the default Anthill3, give it here. The name you provide will be displayed in the logs.
- **Priority.** Optionally, you can have AnthillPro set a priority when the issue is created. Make sure that the priority matches those set in Quality Center.
- **Severity.** Optionally, you can have AnthillPro set a severity when the issue is created. Make sure that the severity matches those set in Quality Center.
- **Status.** If you want to use a different status than the default of new, give it here. Make sure that the status matches those set in Quality Center.

- **Additional Parameters.** Optionally, you can set custom Parameters for the issue. This is entered as a listing of name=value pairs (adheres specifically to the `java.util.Properties` format). Each pair must be included on a separate line. For example:

```
BG_USER_03=SPANIS
          ENGLISH
          FRENCH
          ITALIAN
```

- **Show Environment Variables (optional; advanced).** See Show Environment Variables above.
- **Show Additional Options (advanced).** See Show Additional Options.
- Click **Save**.

Deploy Job with Quality Center Plugin (Issue Tracking) Post-deploy Step

The Resolve Issue step is typically used as part of a post-deployment job that is run after the deploy job, but part of the same workflow. Once the artifacts have been deployed, the Resolve Issue step is run as a separate job in order to ensure that all issues are resolved.

- While each job is different, every job will typically run an Assign Status step; the Resolve Issue step; a Get Changelog step; a Publish Changelog step; and a Create Stamp Step.

1. Go to **Administration**, select the appropriate **project**, and configure the job if not already done.
2. Select the **Insert After** icon of the step prior to the point where the ClearQuest step is to be included. Go to **Issue Tracking > Quality Center Plugin**, select the **Update Quality Center Issue** step, and click **Select**.
 - **Name** the step.
 - **Description.** Provide a short description.
 - **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
 - **QC Server.** Select the correct integration from the drop-down menu (this is the integration set up in the Configure Quality Center Plugin (Issue Tracking) section). If you configured multiple integrations on the Ant-hillPro System page, make sure you select the correct one. Note that it is possible for a single job -- but not a step -- to use different AnthillPro/Quality-Center-server configurations.
 - **Domain Name.** Provide the name of the Quality Center Domain where projects are located. Make sure that this matches those set for the Quality Center testing step.
 - **Project Name.** Give the name of the Quality Center Project where issues are located. Make sure that this matches those set for the Quality Center testing step.
 - **Capture mode.** Choose how to locate bugs to update. If you will list the bug id's, comma separated, such as '4, 24, 13', choose list. If you will match a regex, such as 'QC-([0-9]+)', choose regex.
 - **Issue Id Pattern.** Input a regular expression to locate Quality Center Issue Id within changelog comments.

Add "(" around the portion of the pattern identifying the Quality Center Issue Id (which is simply a number).

The default value, `QC-([0-9]+)` is a regex that parses the change log and applies the attributes of the step to matching issues. For example, if QC-52 was located in the comment of a change, this step would act on issue 52 in the Quality Center database.

- **Assignee.** You must include the Quality Center user this issue must be assigned to. This can be any user with appropriate permissions set in Quality Center.
- **Summary.** Give the Quality Center summary for this issue.
- **Priority.** Optionally, you can have AnthillPro set a priority when the issue is created. Make sure that the priority matches those set in Quality Center.
- **Severity.** Optionally, you can have AnthillPro set a severity when the issue is created. Make sure that the severity matches those set in Quality Center.
- **Status.** If you want to use a different status than the default of new, give it here. Make sure that this matches those set for the Quality Center testing step.
- **Fail Mode.** Select the action you wish AnthillPro to take when an unknown issue ID is encountered.
- **Additional Parameters.** Optionally, you can set custom Parameters for the issue. This is entered as a listing of name=value pairs (adheres specifically to the `java.util.Properties` format). Each pair must be included on a separate line. For example:

```
BG_USER_03=SPANIS
                ENGLISH
                FRENCH
                ITALIAN
```

- **Show Environment Variables (optional; advanced).** See Show Environment Variables above.
- **Show Additional Options (advanced).** See Show Additional Options.

3. Click **Save**.

Create Quality Center Plugin Workflow (Issue Tracking)

Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the Quality Center integration. The process is the same for both jobs that include the Quality Center steps.

1. Go **Administration** and select the appropriate workflow. You can add the job to either your build workflow or as part of a secondary (i.e., testing) workflow.
2. Go to **workflow Main > Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **job** created in the Create Job section, a **job pre-condition script**, and click **Insert Job**.

6. Select the **Properties** tab and add a workflow property. This will ensure that the job runs on the appropriate agent when using a scripted agent filter. Make sure that the property you set here is also configured on the agent.
 - **Name.** Give the name of the Property (the property <name> will be accessed as `${property:<name>}`).
 - **Description.** Provide a description for this Property shown when prompting users for value.
 - **Default Value.** Input the value for this property.
 - **User May Override.** Check if users are able to specify a new value when running this workflow.
 - **Label.** Provide a label for this Property shown when prompting users for value (leave blank to use the Name as the Label).
 - **Is Required.** Check if a non-empty value for this property is required to run workflow.
 - **Allowed Values.** Give the values users are allowed to select for this property (blank for no restriction of value). Separate each value by entering it on its own line.
7. If you created a post-deploy job with the Resolve step, add it after the original job.
8. Click **Save**.

Run Build and View Reports (Quality Center Plugin Issue Tracking)

1. Go to **Dashboard** and select the appropriate workflow.
2. On the **workflow Main** page, click the **Build** button.
3. Once the workflow is complete, select the **Reports** tab.
4. Select a link to view a report.

Quality Center Bug Report

Project: [XPetStore \(SVN\) QC and QTP](#)
 BuildLife: [1096](#)
 Generated On: 03/14/2008 03:00 PM -0400
 Latest Stamp: DEV-701
 Quality Center Server: <http://mtdqg8080/qcbin>

Issue ID: 52

Name: Change the background of my application to red
 Description: Change the background color to red as a demo.
 Status: New
 Assigned To: etm
 Detected By: etm
 Priority:

5. To drill down on each Quality Center step on the Build Life Summary page, see Trace a Build Life to Source.

Quality Center Plugin Function Calls

Following is a list of the function calls used in the Quality Center integration:

TDApi-Ole80.TDConnection	TestSet.StartExecution	TestSet.TSTestFactory	Step.Field
TDConnection.BugFactory	TSScheduler.RunAllLocally	TSTestFactory.NewList	BugFactory.Filter
TDConnection.InitConnectionEx	TSScheduler.TdHostName	TSTest.Name	BugFactory.Item
TDConnection.Login	TSScheduler.Run	TSTest.Type	BugFactory.AddItem
TDConnection.Connect	TSScheduler.ExecutionStatus	TSTest.Status	Filter.Filter
TDConnection.Disconnect	ExecutionStatus.RefreshExecStatusInfo	TSTest.HostName	Filter.NewList
TDConnection.Logout	ExecutionStatus.Finished	TSTest.TestId	Bug.ID
TDConnection.ReleaseConnection	ExecutionStatus.EventsList	TSTest.TestName	Bug.Summary
TDConnection.TestSetTreeManager	ExecEventInfo.EventType	TSTest.LastRun	Bug.Field
TestSetTreeManager.NodeByPath	ExecEventInfo.EventDate	Run.Name	Bug.AssignedTo
TestSetFolder.FindTestSets	ExecEventInfo.EventTime	Run.Status	Bug.DetectedBy
TestSet.Id	TestExecStatus.TestId	Run.Field	Bug.Priority
TestSet.Item	TestExecStatus.TestInstance	Run.StepFactory	Bug.Project
TestSet.Name	TestExecStatus.TsTestId	StepFactory.NewList	Bug.Status
TestSet.TestSetFolder	TestExecStatus.Message	Step.Name	Bug.Post
TestSet.Status	TestExecStatus.Status	Step.Status	

Rally

Create defects, add comments to a defect, update the status of a defect, and report the build success with the Rally 1.05 (backwards compatible) integration. AnthillPro users can also create a report of Rally defects from the changelog.

In order to use the Rally integration, AnthillPro must first be configured with Rally user permissions to create defects, add comments to a defect, update the status of a defect, and report the build success, etc.

The Rally integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab.

Rally Steps:

- **Create Defect.** Creates a new issue in Rally. Typically used during a build job.
- **Add Comments.** Adds Comments from the current changelog to matching Rally defects in the user story. Typic-

ally used during a build job.

- **Change Defect Status.** Updates the status of a Rally defect. Can be used as a post-deployment step added to a deploy workflow to update the state of a Rally defect.
- **Publish Defect Report.** Creates A Report of Rally defects from the current changelog. Typically used during a build job.
- **Report Build Status.** Reports the build status to Rally. Typically used during a build job.

This tutorial will follow a simple project configuration that (a.) uses the add comments, publish defect report, create defect, and report build status steps as part of the build process; (b.) uses the change defect step as part of a post-deployment process; and (c.) makes the Rally defect report and issues available to the AnthillPro UI.

Configuring the Rally Integration

The Rally integration requires a base URL where the Rally server is located as well a user name and password for accessing it.

Rally Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- A project must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See Using Life-Cycle Models.

Configure Rally

All the fields on the Rally Integration page may contain scripts and/or property lookups. See Scripting.

1. Go to **System > Rally** from the Integration menu.
2. On the **Rally Integration** page, click **Edit**.
3. Enter:

- **Rally Base URL.**
- **Issue URL.** You can have AnthillPro automatically generate a link to all of the issues it associates with a Build Life if you give the Issue URL here. Once you give the URL pattern, the issues that appear on the Issues Tab of a Build Life will be linked to the issue in your issue tracker tool for reviewing the issue, adding additional comments, making edits, etc.

Please provide a URL template such as `http://bugs.company.com/browse/${issueId}`. The value `${issueId}` will be replaced in the template with the issue id of the associated issue. This field provides a template which is used throughout AnthillPro to generate links from issues directly to an issue description page within your issue tracker.

- **User Name.** The user name to be used to connect to the Rally server. This name must be configured in Rally in order for the integration to work.
- **Password.** The password used to connect to the Rally server. This password must correspond to the Rally user

name and must be configured in Rally.

- **Password Script (optional).** To use a script or property lookups for the password, leave the Password field blank and enter it here. See Scripting.

4. Click **Set** then **Done**.

Using the Rally Integration

The Rally integration allows AnthillPro users to create defects, add comments to a defect, update the status of a defect, and report the build success in Rally. The integration also allows AnthillPro users to create a report of Rally defects from the changelog.

The example in this tutorial uses Subversion, but the basic configuration is the same for any repository type. Your build job will vary, but the Rally integration is added as a job step similar to what is described below. Though the example goes through the manual creation of a build job, it is possible to use the Job Wizard to create a build job and then manually configure a second job to run as part of the build workflow. For simplicity, the Rally integration steps have been included as part of the build job and part of the post-deployment job in the example. The change defect status is included as part of the post-deployment steps run as part of a deploy workflow.

Using the Rally Integration Prerequisites

- Configuring the Rally Integration must be complete.
- You must have AnthillPro administrative privileges to configure the integration. See [Manage Security](#).
- A project with at least one Build Life must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See [Using Life-Cycle Models](#).

Configure Rally Jobs

Complete job configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the Rally integration.

Build Job with Rally Steps

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.

For this tutorial, the Rally steps are included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery step.

4. **Add Comments.** Select the **Insert After** icon of the step prior to the point where the Rally step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > Rally**, select the **Add Comments** step, and click **Select**.

- **Name.** Provide a name.

- **Description (optional).** Give a description of the step.
- **Issue Id Pattern.** Give a regular expression to locate a Rally defect Id within changelog comments. Add "()" around the portion of the pattern which identifies the actual Issue Id. The pattern "DE[0-9]+" would match (returning the same string as Issue Id) "DE1" and "DE932415", but not match "DEa", "DE", or "DE.1". The pattern "Issue:\\[(DE[0-9]+)\\]" would match "Issue:[DE1]" using Issue Id DE1.

In the example, the pattern includes DE which identifies the issues as belonging to Rally. Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate defect comment in Rally.

- **Additional Comment (optional).** Use this field to add any additional information to the Rally defect.
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
- Click **Save**.

5. **Publish Defect Report.** Select the **Insert After** icon of the **Add Comments** step. Go to **Issue Tracking > Rally**, select the **Publish Defect Report** step, and click **Select**.

- **Name.** Provide a name.
- **Description (optional).** Give a description of the step.
- **Report Name.** Provide a name for the report in AnthillPro (default is the step name).
- **Issue Id Pattern.** Give a regular expression to locate a Rally defect Id within changelog comments. Add "()" around the portion of the pattern which identifies the actual Issue Id. The pattern "DE[0-9]+" would match (returning the same string as Issue Id) "DE1" and "DE932415", but not match "DEa", "DE", or "DE.1". The pattern "Issue:\\[(DE[0-9]+)\\]" would match "Issue:[DE1]" using Issue Id DE1.

In the example, the pattern includes DE identifies the issues as belonging to Rally. Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate defect comment in Rally.

- **Include Dependencies.** Check this field to include change logs from dependencies when searching for issue references.
- **Show Additional Options (advanced).** See Show Additional Options.
- Click **Save**.

6. **Create Defect.** Select the **Insert After** icon of the **Publish Defect Report** step. Go to **Issue Tracking > Rally**, select the **Create Defect** step, and click **Select**. This step will create a defect in Rally on if one of the steps fail.

- **Name.** Provide a name.

- **Description (optional).** Give a description of the step.
 - **Project Key.** Enter the Rally project key to create a general project but not for a specific requirement. Either use this field or the Requirement Key field below.
 - **Requirement Key.** Enter the Rally requirement key to create a defect under a specific Rally requirement. Either use this field or the Project Key field above.
 - **Defect Name.** Give a name for the defect to be created in Rally.
 - **Defect Description.** Give a brief description of the defect. A URL may be included in this field.
 - **Severity.** Select the Rally severity (from the drop-down menu) to be assigned to this step.
 - **Priority.** Select the Rally priority (from the drop-down menu) to be assigned to this step.
 - **Environment.** Select the Rally environment (from the drop-down menu) to be assigned to this step.
 - **State.** Select the Rally state (from the drop-down menu) to be assigned to this step.
 - **Submitted By.** Input the name of the person that submitted the defect. This field must match the AnthillPro Rally user name.
 - **Found in Build.** Identify what build the defect was found in.
 - **Show Additional Options (advanced).** See Show Additional Options.
 - Click **Save**.
7. **Report Build Status.** Select the **Insert After** icon of the step prior to where the Report Build Status is to be included (typically after the Assign Status step). Go to **Issue Tracking > Rally**, select the **Report Build Status** step, and click **Select**.
- To have AnthillPro notify Rally of a successful build, an Assign Status (with the success option selected) step must run prior to adding the Rally-Report Build Status (success) step.
- **Name.** Provide a name.
 - **Description (optional).** Give a description of the step.
 - **Build Definition Id.** Give the Rally Build Definition Id associated with the project status is being assigned to.
 - **Status.** Select the status from the drop-down menu.
 - **Duration (optional).** Identify the duration of the build in milliseconds.
 - **Label.** Provide the Label applied to the build.
 - **Message (optional).** Provide any messages to be sent to Rally.
 - **Show Additional Options (advanced).** See Show Additional Options.
 - Click **Save**.
8. **Report Build Status.** Select the **Insert After** icon of the **Report Build Status (success)** step. Go to **Issue Tracking > Rally**, select the **Report Build Status** step, and click **Select**. To have AnthillPro notify Rally of a failed build, an Assign Status (with the failure option selected) step must run prior to adding the Rally-Report Build Status (failure) step. The Status-Failure step is typically included in the job just after the Status-Success step.

- Configure the Step as in Step 7 (previous), but select **Failed** from the drop-down menu.

Deploy Job with Rally Post-deploy Steps

The Rally-Change Defect Status step is typically used as part of a post-deployment job. Once the artifacts have been deployed (in the example there are two artifact sets: Database and Webapp), the Rally-Change Defect Status step is run as a separate job in order to ensure that all Rally user-story defects are resolved.

- While each job is different, every job will typically run an Assign Status step; the Rally-Change Defect Status step; a Get Changelog step; a Publish Changelog step; and a Create Stamp Step.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** Icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
 - For this tutorial, the Rally-Change Defect Status is included after the Assign Status Step and prior to the other steps.
4. **Change Defect Status**. Select the **Insert After** icon of the step prior to where the Change Defect Step is to be inserted (typically after the Assign Status Step). Go to **Issue Tracking > Rally**, select the **Change Defect Status** step, and click **Select**.
 - **Name**. Provide a name.
 - **Description (optional)**. Give a description.
 - **Defect Key**. The Rally defect key for the state that should be updated.
 - **New State**. Select the Rally state from the drop-down menu.
 - **Show Additional Options (advanced)**. See Show Additional Options.
 - Click **Save**.

Configure Rally Workflows

The Jobs created in the Configure Jobs section must be executed as part of a workflow. This section will assume that an originating workflow and a deploy workflow have already been configured, and will cover the process of adding the jobs to the appropriate workflows. Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the Rally integration.

Build Workflow with Rally Steps

1. Go to **Administration**, select the **project**, and select the **build** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.

5. Select the **Build Job** created in Configure Jobs, a job **pre-condition script**, and click **Insert Job**.

Deploy Workflow with Rally Post-deploy Steps

1. Go to **Administration**, select the **project**, and select the **deploy** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Stop** icon and select **Insert Job After**.
5. Select the **Post Deployment Steps Job** created in the Configure Jobs section, a job **pre-condition script**, and click **Insert Job**.
 - Note that this jobs deploy two artifact sets and that the Post Deploy Steps job will change the Rally defect status for both artifact sets.



Run Workflows and View Reports (Rally)

1. Go to **Dashboard** and select the **workflows** created in the Create Workflows section.
2. On the **Workflow Main** page, click the **Build** button for each workflow.
3. Once the workflows have completed, select the appropriate **Build Life** and click the **Reports** tab.
4. Select a link to view a report. To go to the defect that was generated in Rally, follow the **Defect Id** link.

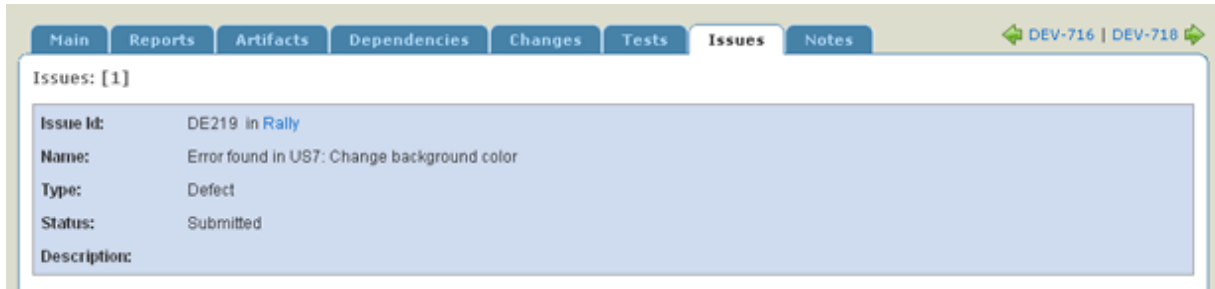
Rally DEFECT REPORT

Project: [XPetStore \(SVN + Rally\)](#)
 BuildLife: [1152](#)
 Generated On: 03/31/2008 04:35 PM -0400
 Latest Stamp: DEV-717
 Rally Server: <https://trial.rallydev.com>

Defect ID: [DE219](#)

Issue Type: Defect
 Name: Error found in US7: Change background color
 Description:
 Status: Submitted

5. Click the **Issues** tab to view the issues AnthillPro created in Rally.



6. To drill down on each Rally step on the **Build Life Summary** page, see Trace a Build Life to Source.

Rally Plugin

Create defects, add comments to a defect, update the status of a defect, and report the build success with the Rally Plugin integration. AnthillPro users can also create a report of Rally defects from the changelog.

In order to use the Rally Plugin integration, AnthillPro must first be configured with Rally user permissions to create defects, add comments to a defect, update the status of a defect, and report the build success, etc.

The integration is written as an AnthillPro Plugin, and expands upon the existing Rally integration to allow for multiple workspaces and tasks. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbanocode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

The Rally Plugin integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab.

Rally Plugin Steps:

- **Create Defect.** Creates a new issue in Rally. Typically used during a build job.

- **Add Comments.** Adds Comments from the current changelog to matching Rally defects in the user story. Typically used during a build job.
- **Change Status.** Updates the status of a Rally defect. Can be used as a post-deployment step added to a deploy workflow to update the state of a Rally defect.
- **Publish Defect Report.** Creates A Report of Rally defects from the current changelog. Typically used during a build job.
- **Report Build Status.** Reports the build status to Rally. Typically used during a build job.

Typical configuration: *(a.)* uses the add comments, publish defect report, create defect, and report build status steps as part of the build process; *(b.)* uses the change defect step as part of a post-deployment process; and *(c.)* makes the Rally defect report and issues available to the AnthillPro UI.

Configuring the Rally Plugin Integration

The Rally Plugin integration requires a base URL where the Rally server is located as well a user name and password for accessing it.

Rally Plugin Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- A project must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See Using Life-Cycle Models.

Configure Rally Plugin

All the fields on the Rally Plugin Integration page may contain scripts and/or property lookups. See Scripting.

The integration is written as an AnthillPro Plugin, and expands upon the existing Rally integration to allow for multiple workspaces and tasks. For older AnthillPro 3.7 versions, you will need to download the integration from Supportal [<http://support.urbancode.com/>] and then upload it to the server. Once uploaded, ensure the Plugin is active.

1. Go to **System > Rally Plugin** from the Integration menu.
2. On the **Rally Plugin** page, click **Create New**.
3. Enter:
 - **Name.** Give a name to this integration. This name will be used by the AnthillPro system. If you configure more than one integration, make sure to select the correct configuration during job creation.
 - **Rally Base URL.** Provide the Rally server URL. Example: <https://trial.rallydev.com> (do not use a trailing slash).
 - **Rally Version.** Give the Rally server Webservices Version your Rally server supports. Example: 1.16 (the version must be an exact match for the integration to work). If you have multiple versions of Rally, you will need to configure a new integration for each version.

- **Rally Workspace Name.** Provide the Rally workspace to use for this integration. Each configured integration supports only one workspace. If you want AnthillPro to participate in multiple workspaces, configure a separate integration for each.
- **User Name.** The user name to be used to connect to the Rally server. This name must be configured in Rally in order for the integration to work.
- **Password.** The password used to connect to the Rally server. This password must correspond to the Rally user name and must be configured in Rally.
 - Confirm password.
- **Password Script (optional).** To use a script or property lookups for the password, leave the Password field blank and enter it here. See Scripting.

4. Click **Set** then **Done**.

Using the Rally Plugin Integration

The Rally Plugin integration allows AnthillPro users to create defects, add comments to a defect, update the status of a defect, and report the build success in Rally. The integration also allows AnthillPro users to create a report of Rally defects from the changelog.

Your build job will vary, but the Rally integration is added as a job step similar to what is described below. Though the example goes through the manual creation of a build job, it is possible to use the Job Wizard to create a build job and then manually configure a second job to run as part of the build workflow. For simplicity, the Rally integration steps have been included as part of the build job and part of the post-deployment job in the example. The change defect status is included as part of the post-deployment steps run as part of a deploy workflow.

Using the Rally Plugin Integration Prerequisites

- Configuring the Rally Plugin Integration must be complete.
- You must have AnthillPro administrative privileges to configure the integration. See [Manage Security](#).
- A project with at least one Build Life must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See [Using Life-Cycle Models](#).

Configure Rally Plugin Jobs

Complete job configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the Rally Plugin integration.

Build Job with Rally Plugin Steps

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.

The Rally steps are typically included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery step.

4. **Add Comments.** Select the **Insert After** icon of the step prior to the point where the Rally step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > Rally Plugin**, select the **Add Comments** step, and click **Select**.

- **Name.** Provide a name.
- **Description (optional).** Give a description of the step.
- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **Defect Id Pattern.** Give a regular expression to locate a Rally defect Id within changelog comments. Add "(" around the portion of the pattern which identifies the actual Issue Id. The pattern "DE[0-9]+" would match (returning the same string as Issue Id) "DE1" and "DE932415", but not match "DEa", "DE", or "DE.1". The pattern "Issue:\\((DE[0-9]+)\\)" would match "Issue:[DE1]" using Issue Id DE1.

In the example, the pattern includes DE which identifies the issues as belonging to Rally. Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate defect comment in Rally.

- **Task Id Pattern.** Give a regular expression to locate a Rally defect Id within changelog comments. Add "(" around the portion of the pattern which identifies the actual task Id. The pattern 'TA[0-9]+' would match (returning the same string as task ids) 'TA1' and 'TA932415', but not match 'TAa', 'TA', or 'TA.1' The pattern 'Issue:\\((TA[0-9]+)\\)' would match 'Issue:[TA1]' using task ID TA1.

In the example, the pattern includes TA which identifies the issues as belonging to Rally. Whenever AnthillPro comes across this regular expression, it will automatically add the comment to the appropriate task in Rally.

- **User Story Id Pattern.** Give a regular expression to locate a Rally defect Id within changelog comments. Add "(" around the portion of the pattern which identifies the actual user story Id. The pattern 'US[0-9]+' would match (returning the same string as user story ids) 'US1' and 'US932415', but not match 'USa', 'US', or 'US.1' The pattern 'Issue:\\((US[0-9]+)\\)' would match 'Issue:[US1]' using user story ID US1.

In the example, the pattern includes US which identifies the issues as belonging to Rally. Whenever AnthillPro comes across this regular expression, it will automatically add the comment to the appropriate user story in Rally.

- **Additional Comment (optional).** Use this field to add any additional information to the Rally defect.
- **Rally Workspace.** Select the workspace you configured in the previous section.
- **Show Environment Variables (optional; advanced).** Give any optional environment variables in "name=value" format.

Environment variables values may contain references to existing values in the following format: name=\${env/<NAME>} ;value. If the value of the <NAME> variable is "value2" in the current environment, then the above example will be expanded to: name=value2 ;value.

Using this technique, it is possible add an entry to PATH in the following manner: PATH=my/path/entry;0. Case is significant even on Windows systems.

- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to

continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.

- **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
- **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
- **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
- Click **Save**.

5. **Publish Defect Report.** Select the **Insert After** icon of the **Add Comments** step. Go to **Issue Tracking > Rally Plugin**, select the **Publish Defect Report** step, and click **Select**.

- **Name.** Provide a name.
- **Description (optional).** Give a description of the step.
- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **Report Name.** Provide a name for the report in AnthillPro (default is the step name).
- **Defect Id Pattern.** Give a regular expression to locate a Rally defect Id within changelog comments. Add "()" around the portion of the pattern which identifies the actual Issue Id. The pattern "DE[0-9]+" would match (returning the same string as Issue Id) "DE1" and "DE932415", but not match "DEa", "DE", or "DE.1". The pattern "Issue:\\[(DE[0-9]+)\\]" would match "Issue:[DE1]" using Issue Id DE1.

In the example, the pattern includes DE which identifies the issues as belonging to Rally. Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate defect comment in Rally.

- **Task Id Pattern.** Give a regular expression to locate a Rally defect Id within changelog comments. Add "()" around the portion of the pattern which identifies the actual task Id. The pattern 'TA[0-9]+' would match (returning the same string as task ids) 'TA1' and 'TA932415', but not match 'TAa', 'TA', or 'TA.1' The pattern 'Issue:\\[(TA[0-9]+)\\]' would match 'Issue:[TA1]' using task ID TA1.

In the example, the pattern includes TA, which identifies the issues as belonging to Rally. Whenever AnthillPro comes across this regular expression, it will automatically add the comment to the appropriate task in Rally.

- **User Story Id Pattern.** Give a regular expression to locate a Rally defect Id within changelog comments. Add "()" around the portion of the pattern which identifies the actual user story Id. The pattern 'US[0-9]+' would match (returning the same string as user story ids) 'US1' and 'US932415', but not match 'USa', 'US', or 'US.1' The pattern 'Issue:\\[(US[0-9]+)\\]' would match 'Issue:[US1]' using user story ID US1.

In the example, the pattern includes US which identifies the issues as belonging to Rally. Whenever AnthillPro comes across this regular expression, it will automatically add the comment to the appropriate user story in Rally.

- **Rally Workspace.** Select the workspace you configured in the previous section.
- **Show Environment Variables (advanced).** See Show Environment Variables above.
- **Show Additional Options (advanced).** See Show Additional Options above.

- Click **Save**.

6. **Create Defect.** Select the **Insert After** icon of the **Publish Defect Report** step. Go to **Issue Tracking > Rally Plugin**, select the **Create Defect** step, and click **Select**. This step will create a defect in Rally on if one of the steps fail.

- **Name.** Provide a name.
- **Description (optional).** Give a description of the step.
- **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
- **Project Name.** Enter the Rally project key to create a general project but not for a specific requirement. Either use this field or the Requirement Key field below.

If you do not specify a project name, you must give the user story Id below -- but not both.

- **User Story Id Pattern.** Give a regular expression to locate a Rally defect Id within changelog comments.

If you do not specify a user story Id, you must give a project name above -- but not both.

Add "()" around the portion of the pattern which identifies the actual user story Id. The pattern 'US[0-9]+' would match (returning the same string as user story ids) 'US1' and 'US932415', but not match 'USa', 'US', or 'US.1' The pattern 'Issue:\\[(US[0-9]+)\\]' would match 'Issue:[US1]' using user story ID US1.

In the example, the pattern includes US which identifies the issues as belonging to Rally. Whenever AnthillPro comes across this regular expression, it will automatically add the comment to the appropriate user story in Rally.

- **Defect Name.** Give a name for the defect to be created in Rally.
- **Defect Description.** Give a brief description of the defect. A URL may be included in this field.
- **Severity.** Provide the severity, if desired, that this defect should be reported as. What is input here must match exactly the severity as defined in Rally.
- **Priority.** Give the priority, if desired, that this defect should be reported as. What is input here must match exactly the priority as defined in Rally.
- **Environment.** Give the environment, if desired, that this defect should be reported as. What is input here must match exactly the environment as defined in Rally.
- **State.** Give the state, if desired, that this defect should be reported as. What is input here must match exactly the state as defined in Rally.
- **Submitted By.** Input the name of the person that submitted the defect. This field must match a Rally user name (e.g., AnthillPro).
- **Rally Workspace.** Select the workspace you configured in the previous section.
- **Show Environment Variables (advanced).** See Show Environment Variables above.
- **Show Additional Options (advanced).** See Show Additional Options above.
- Click **Save**.

7. **Report Build Status.** Select the **Insert After** icon of the step prior to where the Report Build Status is to be included (typically after the Assign Status step). Go to **Issue Tracking > Rally Plugin**, select the **Report Build**

Status step, and click **Select**.

To have AnthillPro notify Rally of a successful build, an Assign Status (with the success option selected) step must run prior to adding the Rally-Report Build Status (success) step.

- **Name.** Provide a name.
 - **Description (optional).** Give a description of the step.
 - **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
 - **Build Definition Id.** Give the Rally Build Definition Id associated with the project status is being assigned to.
 - **Status.** Select the status from the drop-down menu. Select "success" from the drop-down. To have AnthillPro also report on failed builds, see *Item 8* below.
 - **Duration (optional).** Identify the duration of the build in milliseconds.
 - **Label.** Provide the Label applied to the build.
 - **Message (optional).** Provide any messages to be sent to Rally.
 - **Rally Workspace.** Select the workspace you configured in the previous section.
 - **Show Environment Variables (advanced).** See Show Environment Variables above.
 - **Show Additional Options (advanced).** See Show Additional Options above.
 - Click **Save**.
8. **Report Build Status (failure).** Select the **Insert After** icon of the **Report Build Status** step. Go to **Issue Tracking > Rally Plugin**, select the **Report Build Status** step, and click **Select**. To have AnthillPro notify Rally of a failed build, an Assign Status (with the failure option selected) step must run prior to adding the Rally-Report Build Status (failure) step. The Status-Failure step is typically included in the job just after the Status-Success step.
- Configure the Step as in Step 7 (previous), but select **Failed** from the drop-down menu.

Rally Plugin Post-deploy Steps

The Rally-Change Defect Status step is typically used as part of a post-deployment job. Once the artifacts have been deployed (in the example there are two artifact sets: Database and Webapp), the Rally-Change Defect Status step is run as a separate job in order to ensure that all Rally user-story defects are resolved.

- While each job is different, every job will typically run an Assign Status step; the Rally-Change Defect Status step; a Get Changelog step; a Publish Changelog step; and a Create Stamp Step.
1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** Icon.
 2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
 3. Follow the steps for creating a build job.
 - For this tutorial, the Rally-Change Defect Status is included after the Assign Status Step and prior to the other steps.

4. **Change Defect Status.** Select the **Insert After** icon of the step prior to where the Change Defect Step is to be inserted (typically after the Assign Status Step). Go to **Issue Tracking > Rally Plugin**, select the **Change Defect Status** step, and click **Select**.
 - **Name.** Provide a name.
 - **Description (optional).** Give a description.
 - **Working Directory Offset (optional).** Give the working directory to use when executing this command. This is relative to current working directory. To use the current directory, leave this field blank.
 - **Defect Id.** Give a comma-separated list of IDs of the defects or tasks for which the state should be updated. For example: Use `${gvy:def result='' ; IssueHelper.getIssueArray().each{it -> result += it.issueId + ", "\} ; return result}` to update all issues in the current Build Life.
 - **Object Type.** Select the correct object type you want this step to apply in Rally. Choose either Defect or Task.
 - **New State.** Provide the Rally state. What is given here must exactly match the state as defined in Rally.
 - **Rally Workspace.** Select the workspace you configured in the previous section.
 - **Show Environment Variables (advanced).** See Show Environment Variables above.
 - **Show Additional Options (advanced).** See Show Additional Options above.
 - Click **Save**.

Configure Rally Plugin Workflows

The Jobs created in the Configure Jobs section must be executed as part of a workflow. This section will assume that an originating workflow and a deploy workflow have already been configured, and will cover the process of adding the jobs to the appropriate workflows. Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the Rally integration.

Build Workflow with Rally Plugin Steps

1. Go to **Administration**, select the **project**, and select the **build** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Build Job** created in Configure Jobs, a job **pre-condition script**, and click **Insert Job**.

Deploy Workflow with Rally Plugin Post-deploy Steps

1. Go to **Administration**, select the **project**, and select the **deploy** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Stop** icon and select **Insert Job After**.

5. Select the **Post Deployment Steps Job** created in the Configure Jobs section, a job **pre-condition script**, and click **Insert Job**.
 - Note that this jobs deploy two artifact sets and that the Post Deploy Steps job will change the Rally defect status for both artifact sets.

Run Workflows and View Reports (Rally Plugin)

1. Go to **Dashboard** and select the **workflows** created in the Create Workflows section.
2. On the **Workflow Main** page, click the **Build** button for each workflow.
3. Once the workflows have completed, select the appropriate **Build Life** and click the **Reports** tab.
4. Select a link to view a report. To go to the defect that was generated in Rally, follow the **Defect Id** link.
5. Click the **Issues** tab to view the issues AnthillPro created in Rally.
6. To drill down on each Rally step on the **Build Life Summary** page, see Trace a Build Life to Source.

TeamTrack

Create a new defect, add comments to a defect, and transition a defect with the TeamTrack integration. AnthillPro users can also create a report of TeamTrack defects from the changelog.

In order to use the integration, AnthillPro must first be configured with TeamTrack. The integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab. See Using the TeamTrack Integration.

TeamTrack Steps:

- **Add Comments.** Add Comments from the current changelog to matching TeamTrack defects. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary.
- **Create Defect.** Create a new defect in a TeamTrack. Typically used during a secondary process in combination with the Transition Defect step.
- **Publish Defect Report.** Create a Report of TeamTrack defects from the current changelog. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary. Typically used in combination with the Add Comments step.
- **Transition Defect.** Transition a defect in a TeamTrack. Typically used during a secondary process in combination with the Create Defect step. The Issue Key field of this step should include at least one property specified on the corresponding workflow. See Defects Workflow with TeamTrack Steps.

This tutorial will follow a simple project configuration that uses the Add Comments and Publish Defect Report steps as part of a build workflow. The Create Defect and Transition Defect steps are used as part of a secondary (non-originating) workflow. The example in this tutorial uses Maven, but the basic configuration is similar for any repository type. Your jobs will vary, but the TeamTrack integration is added as job steps similar to what is described below. Though the example goes through the manual creation of a build job, it is possible to use the Job Wizard to create a build job and then manually configure a second job to run as part of the same workflow.

Configure TeamTrack

Let AnthillPro know about TeamTrack. Any steps within AnthillPro relying on TeamTrack will not work until this integration is configured. These fields may all contain scripts and/or property lookups. See Scripting.

TeamTrack Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- A project must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See Using Life-Cycle Models.
- AnthillPro must be set up as a TeamTrack user. See TeamTrack documentation.

TeamTrack Configuration

1. Go to **System > TeamTrack** from the Integrations menu.
2. On the **TeamTrack Integration** page, click **Edit**.
3. Configure the integration:
 - **TeamTrack Server URL.** Provide the TeamTrack server URL.
 - **Issue URL.** You can have AnthillPro automatically generate a link to all of the issues it associates with a Build Life if you give the Issue URL here. Once you give the URL pattern, the issues that appear on the Issues Tab of a Build Life will be linked to the issue in your issue tracker tool for reviewing the issue, adding additional comments, making edits, etc.

Please provide a URL template such as `http://bugs.company.com/browse/${issueId}`. The value `${issueId}` will be replaced in the template with the issue id of the associated issue. This field provides a template which is used throughout AnthillPro to generate links from issues directly to an issue description page within your issue tracker.
 - **User Name.** Give the user name to be used to connect to the TeamTrack server.
 - **Password.** Provide the password to be used to connect to the TeamTrack server.
 - **Password Script.** To use a script or property lookups for the password, leave the Password field blank and enter it here. See Scripting.
 - **Use Basic Authentication.** Check the box to enable basic authentication. By Default, AnthillPro passes the user name password as an argument on the SOAP request.
4. Click **Set** then **Done**.

Using the TeamTrack Integration

This tutorial adds a TeamTrack comment and publishes a TeamTrack report as part of the build workflow. Complete job configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the TeamTrack integration.

Using the TeamTrack Integration Prerequisites

- The Configure TeamTrack section of this tutorial must be complete.
- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- A project with at least one Build Life must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See Using Life-Cycle Models.

Configure TeamTrack Jobs

This step will follow a job configuration that adds a comment, publishes a report, creates a defect, and transitions a defect in TeamTrack.

Build Job with TeamTrack Steps

The Add Comments and Publish Defect Report steps are included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the typical build job.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
 2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
 3. Follow the steps for creating a build job.
 4. **Add Comments.** Select the **Insert After** icon of the step prior to the point where the TeamTrack step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > TeamTrack**, select the **Add Comments** step, and click **Select**.
 - **Name** the step.
 - **Description.** Provide a short description.
 - **Issue Id Pattern.** Give a regular expression to locate a TeamTrack defect Id within changelog comments. Add "()" around the portion of the pattern which identifies the actual Issue Id. The pattern "TST-[0-9]+" would match (returning the same string as Issue Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:[(TST-[0-9]+)]" would match "Issue:[TST-1]" using issue Id TST-1.

In the example below, the pattern (?:(BUG|CHG|ENH|PRB|TSK)([0-9]+) identifies the issue as belonging to a TeamTrack project. Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate comment in TeamTrack.
- **Title.** Provide the TeamTrack title.
 - **Additional Comment.** Give any information to be added to the comment (in addition to the commit comment).
 - **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.

- **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
- **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
- **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

5. Click **Save**.

6. **Publish Defect Report.** Select the **Insert After** icon of the **Add Comments** step. Go to **Issue Tracking > TeamTrack**, select the **Publish Defect Report** step, and click **Select**.

- **Name** the step.
- **Description.** Provide a short description.
- **Report Name.** Give the name for this report (default is same as step name).
- **Issue Id Pattern.** Give a regular expression to locate a TeamTrack defect Id within changelog comments. Add "(" around the portion of the pattern which identifies the actual Issue Id. The pattern "TST-[0-9]+" would match (returning the same string as Issue Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Bug:\[(TST-[0-9]+)\]" would match "Bug:[TST-1]" using Bug Id TST-1.

In the example below, the pattern (?:(BUG|CHG|ENH|ORB|TSK)([0-9]+) identifies the issue as belonging to a TeamTrack project. Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate comment in TeamTrack.

- **Include Dependencies.** Check the box to include the change logs from dependencies when searching for issue references.
- **Show Additional Options.** See Show Additional Options.

7. Click **Save**.

Defects Job with TeamTrack Steps

The Create Defect and Transition Defect steps are typically added to a job that run as part of a secondary, or non-originating, workflow. To ensure that the proper defects are transitioned, the Issue Key field of the **Transition Defect** step should include at least one property.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. Select the **Insert After** icon of the step prior to the point where the TeamTrack step is to be included. Go to **Issue Tracking > TeamTrack**, select the **Create Defect** step, and click **Select**.
 - **Name** the step.
 - **Description.** Provide a short description.

- **Project Name.** Give the TeamTrack project name that the defect belongs to. *If not specified, the Fully Qualified Project Name must be specified.*
- **Fully Qualified Project Name.** Provide the Fully Qualified Project Name which includes the Folder Path separated by "|". *If not specified, the Project Name must be specified.*
- **Issue Type.** Select the TeamTrack issue type from the drop-down menu. Choose Bug Report, Change Request, Enhancement Request, Problem Report, or Task. See TeamTrack documentation for more on issue types.
- **Title.** Give the title for this defect.
- **Issue Description.** Provide a description for this defect.
- **Functional Area.** Give the Functional Area the defect appears in. *Default value is Administrator.*
- **How Found.** Tell how the defect was found in the code base. *Default value is Automated Test.*
- **Severity.** Provide the Severity of the defect. *Default value is Low.*
- **Detected By.** Give the user the defect was detected by. *Default is the user set up in the TeamTrack server configuration.*
- **State.** Provide the State for the defect. *Default value is New.*
- **Show Additional Options.** See Show Additional Options.

5. Click **Save**.

6. **Transition Defect.** Select the **Insert After** icon of the **Create Defect** step. Go to **Issue Tracking > TeamTrack**, select the **Transition Defect** step, and click **Select**.

- **Name** the step.
- **Description.** Provide a short description.
- **Issue Key.** Give the issue key to be transitioned in TeamTrack. This will typically be a property to ensure that the correct issue is transitioned. See also Defects Workflow with TeamTrack Steps.
- **Transition Name.** Provide the name of the Transition to perform in TeamTrack.
- **Additional Options.** Give any additional fields that are required to perform a transition. These fields can be entered on per line in **name=value** format, where name is the name of the property and value is the value that will be passed to TeamTrack.
- **Show Additional Options.** See Show Additional Options.

7. Click **Save**.

Add TeamTrack Jobs to Workflows

The Jobs created in the Configure Jobs section must be executed as part of a workflow. This section will assume that both an originating workflow and non-originating (secondary) workflow have already been configured. The topics covered in detail below are specific to using the TeamTrack integration. Complete workflow configuration is beyond the scope of this tutorial.

Build Workflow with TeamTrack Steps

1. Go to **Administration**, select the **project**, and select the **build** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Build** job created in the Configure Jobs section, a **job pre-condition** script, and click **Insert Job**.

Defects Workflow with TeamTrack Steps

To ensure that the proper defects are transitioned, the Defects workflow will typically include a property to ensure that the correct issue is transitioned. The property specified on the workflow must correspond to the Issue Key field of the **Transition Defect** job step. See also Defects Job with TeamTrack Steps.

1. Go to **Administration**, select the **project**, and select the **Defects** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Defects Job** created in the Configure Jobs section, a **job pre-condition** script, and click **Insert Job**.
6. Go to **Properties > New Property** to create a workflow property. A property must be set to ensure that the correct issue is transitioned, and must correspond to the Issue Key field of the **Transition Defect** job step. See also Defects Job with TeamTrack Steps.
 - **Name** the property.
 - **Description.** Give a description of the property.
 - **Default Value.** Provide the value of the property.
 - **User May Override.** Check the box to allow users to specify a new value when running the workflow.
 - **Label.** Give the label for the property that will be shown when a user is prompted for a value. *If no Label is specified, the property name will be used.*
 - **Is Required.** Check the box if a non-empty value for this property is required to run the workflow.
 - **Allowed Values.** Provide the values users are allowed to select for this property.
 - Click **Save**.

Run Workflows and View Reports (TeamTrack)

1. Go to **Dashboard** and select the **workflow** created in the Add Jobs to Workflow section.
2. On the workflow **Main** page, click the **Build** button for the workflow.
 - This procedure is the same for both workflows.

3. Once the **workflow** has completed, select the appropriate **Build Life** and click the **Reports** tab.
4. Select a **link** to view a report.

TeamTrack DEFECT REPORT

Project: [Maven-Example \(TeamTrack\)](#)
BuildLife: [431](#)
Generated On: 05/19/2008 01:51 PM -0400
Latest Stamp: 48
TeamTrack Server: <http://evalteamtrack/>

Defect ID: PRB00101

Issue Type: Problem Report
Name: Anthill3 Creted Problem Report
Description: This is a problem Created by Anthill3.
Status: Triage

5. Click the **Issues** tab to view the issues AnthillPro created in TeamTrack.



6. To drill down on each TeamTrack step on the Build Life Summary page, see Trace a Build Life to Source.

Team Foundation Server (TFS) Project Tracking

The TFS integration is implemented as AnthillPro job steps configured on the Job Configuration page. Currently,

AnthillPro enables users to create a new WorkItem, add comments to a WorkItem, publish a WorkItem report, and resolve WorkItems with the TFS integration. To use the TFS integration, AnthillPro must first be configured to run the appropriate job steps as well as configured with TFS source control (see Team Foundation Server (TFS) Source Control).

TFS Steps:

- **Create WorkItem.** Create a new TFS WorkItem when this job runs. Typically used during a build job. See Build Job with TFS Steps.
- **Add Comments.** Add comments from the current changelog to matching TFS issues. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary. When used, the step will automatically look for WorkItems referenced in source changes. You may additionally configure it to search source change comments. See Build Job with TFS Steps.
- **Publish WorkItem Report.** Create a Report of TFS WorkItems from the current changelog. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary (as part of the job). The step will automatically look for WorkItems referenced in source changes, and may be configure to search source change comments. See Build Job with TFS Steps.
- **Resolve WorkItem.** Used to resolve or close a TFS item. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary. You can also have AnthillPro pass a comment at the time the issue is updated. Can be used as a post-deployment step added to a deploy workflow to update the state of a TFS item. See Deploy Job with TFS Resolve WorkItem Step.

When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab (see Add TFS Jobs to Workflows). Your jobs will vary, but the TFS integration is used similar to what is described below.

TFS Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- The TFS repository must first be configured with AnthillPro. A project must be active in AnthillPro that user TFS for source control. See Team Foundation Server (TFS) Source Control.
- The TFS client must be installed on the same machine as the AnthillPro agent that will be responsible for running the TFS job steps (commands).
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See Using Life-Cycle Models.

Configure TFS Integration

Let AnthillPro know about the TFS integration. Any job steps within AnthillPro relying on TFS will not work until this integration is configured.

1. Go to **System > TFS** from the Integrations menu.
2. On the **TFS Integration** page, click **Edit**.
3. Give the TFS server URL provided during source configuration. See Team Foundation Server (TFS) Source Control.

- **Issue URL.** You can have AnthillPro automatically generate a link to all of the issues it associates with a Build Life if you give the Issue URL here. Once you give the URL pattern, the issues that appear on the Issues Tab of a Build Life will be linked to the issue in your issue tracker tool for reviewing the issue, adding additional comments, making edits, etc.

Please provide a URL template such as `http://bugs.company.com/browse/${issueId}`. The value `${issueId}` will be replaced in the template with the issue id of the associated issue. This field provides a template which is used throughout AnthillPro to generate links from issues directly to an issue description page within your issue tracker.

4. Click **Set** then **Done**.
5. Proceed to Using the TFS Integration to add TFS steps to a job.

Using the TFS Integration

Two jobs need to be created to use all the TFS integration steps: (a.) A build job with the TFS Add Comment, Publish WorkItem Report, and Create WorkItem steps. (b.) A post-deployment job that includes the Resolve WorkItem step. See Build Job with TFS Steps and Deploy Job with TFS Resolve WorkItem Step.

Before continuing, make sure the TFS repository has been configured with AnthillPro, and that the TFS integration has been configured. See Team Foundation Server (TFS) Source Control and Configure TFS Integration.

Build Job with TFS Steps

The Add Comments, Create Issue, and Publish Issue Report steps are included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the typical Build job.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. Follow the steps for creating a build job.
3. **Add Comments.** Select the **Insert After** icon of the step prior to the point where the TFS step is to be included (typically the Artifact Delivery step). Go to **Issue Tracking > Team Foundation Server**, select the **Add Comments** step, and click **Select**.

- **Name** the step.
- **Description.** Provide a short description.
- **WorkItem Id Pattern.** Give a regular expression to locate a TFS defect Id within changelog comments. Add "(" around the portion of the pattern which identifies the actual WorkItem Id. The pattern "TST-[0-9]+" would match (returning the same string as WorkItem Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:\\[(TST-[0-9]+)\\]" would match "Issue:[TST-1]" using WorkItem Id TST-1.

Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate comment in TFS.

- **Additional Comment.** Give any information to be added to the TFS comment (in addition to the commit comment).
- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.

- **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
 - **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.
 - Click **Save**.
4. **Create WorkItem.** Select the **Insert After** icon of the **Add Comments** step. Go to **Issue Tracking > Team Foundation Server**, select the **Create WorkItem** step, and click **Select**. This step will create an issue if a failure occurs.
- **Name** the step.
 - **Description.** Provide a short description.
 - **Project Key.** Enter the TFS project key to create the WorkItem under.
 - **Summary.** Enter the WorkItem summary.
 - **Type.** From the drop-down menu, select the type of WorkItem AnthillPro is to create. Currently, AnthillPro can automatically create either a Bug or Task Work Item.
 - **Environment.** Give the environment for the issue.
 - **Issue Description.** Enter the description.
 - **Assignee.** Provide the assignee for the new issue. An assignee of -1 will use TFS's automatic assignment.
 - **Show Additional Options.** See Show Additional Options.
 - Click **Save**.
5. **Publish WorkItem Report.** Select the **Insert After** icon of the **Create Issue** step. Go to **Issue Tracking > Team Foundation Server**, select the **Publish WorkItem Report** step, and click **Select**.
- **Name** the step.
 - **Description.** Provide a short description.
 - **Report Name.** If a new name is not given, the step name will be used.
 - **WorkItem Id Pattern.** Give a regular expression to locate a TFS defect Id within changelog comments. Add "(" around the portion of the pattern which identifies the actual WorkItem Id. The pattern "TST-[0-9]+" would match (returning the same string as WorkItem Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:\\[(TST-[0-9]+)\\]" would match "Issue:[TST-1]" using WorkItem Id TST-1.
- Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate comment in TFS.
- **Include Dependencies.** Check to include change logs from dependencies when searching for issues.

- **Show Additional Options.** See Show Additional Options.
6. Click **Save**.
 7. To have AnthillPro automatically resolve Work Items, Proceed to Deploy Job with TFS Resolve WorkItem Step. If not using the resolve step, see Add TFS Jobs to Workflows.

Deploy Job with TFS Resolve WorkItem Step

The Resolve WorkItem step is typically used as part of a post-deployment job. Once the artifacts have been deployed, the Resolve WorkItem step can be run as a separate job to ensure that all TFS WorkItems are resolved.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. Follow the steps for creating a build job.
3. Select the **Insert After** icon of the step prior to the point where the TFS step is to be included. Go to **Issue Tracking > Team Foundation Server**, select the **Resolve WorkItem** step, and click **Select**.
 - **Name** the step.
 - **Description.** Provide a short description.
 - **WorkItem ID.** Give the name of the WorkItem to be resolved.
 - **State.** Give the state to set the WorkItem to.
 - **Reason.** Give the reason for the WorkItem state change.
 - **Show Additional Options.** See Show Additional Options.
 - Click **Save**.

Add TFS Jobs to Workflows

The Jobs created in the Using the TFS Integration section must be executed as part of a workflow. This section will assume that an originating workflow has already been configured, and will cover the process of adding the TFS job(s) to the appropriate workflow(s). The topics covered in detail below are specific to using the TFS integration.

Build Workflow with TFS Steps

1. Go to **Administration**, select the **project**, and select the **build** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Build** job created in the Using the TFS Integration section, a **job pre-condition** script, and click **Insert Job**.

Deploy Workflow with TFS Post-deploy Step

1. Go to **Administration**, select the **project**, and select the **deploy** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Post Deployment Steps** job created in the Using the TFS Integration section, a **job pre-condition** script, and click **Insert Job**.

Run TFS Workflows and View Reports

1. Go to **Dashboard** and select the appropriate **workflow**.
2. On the workflow **Main** page, click the **Build** button for the workflow.
 - This procedure is the same for the build and deploy workflows.
3. Once the **workflow** has completed, select the appropriate **Build Life** and click the **Reports** tab.
4. Select a **link** to view a report. To go to the WorkItem that was generated in TFS, follow the link.
5. Click the **Issues** tab to view the WorkItems AnthillPro created in TFS.
6. To drill down on each TFS step on the Build Life Summary page, see Trace a Build Life to Source.

VersionOne

Create a new defect, add comments to an defect, and resolve defects with the VersionOne integration. AnthillPro users can also create a report of VersionOne defects from the changelog.

In order to use the VersionOne integration, AnthillPro must first be configured to run the appropriate steps within VersionOne. The VersionOne integration is implemented as AnthillPro job steps configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab.

VersionOne Steps:

- **Create Defect.** Create a new defect in a VersionOne. Typically used during a build job.
- **Add Comments.** Add comments from the current changelog to matching VersionOne defects. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary.
- **Resolve Defect.** Resolve VersionOne defect. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary. Can be used as a post-deployment step added to a deploy workflow to update the state of a VersionOne defect.
- **Publisher Defect Report.** Create a Report of VersionOne defects from the current changelog. In order to use this step, a set working directory, get changelog, and publish changelog step is necessary.

This tutorial will follow a simple project configuration using the VersionOne integration steps. The example in this tutorial uses Subversion, but the basic configuration is similar for any repository type. Your jobs will vary, but the VersionOne integration is added as a job step similar to what is described below. The example goes through the manual creation of a build job; however, it is possible to use the Job Wizard to create a build job and then manually configure a second job to run as part of the same workflow.

Configure VersionOne

Let AnthillPro know about VersionOne. Any steps within AnthillPro relying on VersionOne will not work until this integration is configured. The VersionOne integration requires the URL where the VersionOne server is located as well as a user name and password for accessing it. These fields may all contain scripts and/or property lookups. See Scripting.

VersionOne Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- A project must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See Using Life-Cycle Models.

Configure VersionOne

1. Go to **System > VersionOne** from the Integrations menu.
 2. On the **VersionOne Integration** page, click **Edit**.
 3. Configure the integration:
 - **VersionOne Server URL.** Input the VersionOne server URL.
 - **Issue URL.** You can have AnthillPro automatically generate a link to all of the issues it associates with a Build Life if you give the Issue URL here. Once you give the URL pattern, the issues that appear on the Issues Tab of a Build Life will be linked to the issue in your issue tracker tool for reviewing the issue, adding additional comments, making edits, etc.

Please provide a URL template such as `http://bugs.company.com/browse/${issueId}`. The value `${issueId}` will be replaced in the template with the issue id of the associated issue. This field provides a template which is used throughout AnthillPro to generate links from issues directly to an issue description page within your issue tracker.
 4. Click **Set** then **Done**.
- **User Name.** Give the user name to be used to connect to the VersionOne server.
 - **Password.** Provide and confirm the password to be used to connect to the VersionOne server.
 - **Password Script.** To use a script or property lookups for the password, leave the Password field blank and enter it here. See Scripting.

Using the VersionOne Integration

This tutorial will follow a job configuration that adds a VersionOne comment, publishes a VersionOne report, and creates a defect as part of a build workflow. The resolve defect step is added as a post-deployment step to a deploy workflow.

Using the VersionOne Integration Prerequisites

- Configure VersionOne must be complete.
- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.
- A project with at least one Build Life must be active in AnthillPro.
- A Life-Cycle Model must be configured with the appropriate Status and Artifact Sets. See Using Life-Cycle Models.

Configure VersionOne Jobs

Two jobs need to be created to use all the VersionOne integration steps. (a.) A build job is created with the Add Comment, Publish Defect Report, and Creates Defect steps. (b.) The Resolve Defect step is added as a post-deployment job.

Build Job with VersionOne Steps

The Add Comments, Create Defect, and Publish Defect Report steps are included after the Populate Workspace, Changelog, Stamp, Dependency, Build, Publish Changelog, and Artifact Delivery steps of the Build job.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
 2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
 3. Follow the steps for creating a build job.
 4. **Add Comments.** Select the **Insert After** icon of the step prior to the point where the VersionOne step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > VersionOne**, select the **Add Comments** step, and click **Select**.
 - **Name** the step.
 - **Description.** Provide a short description.
 - **Issue Id Pattern.** Give a regular expression to locate a VersionOne defect Id within changelog comments. Add "()" around the portion of the pattern which identifies the actual Issue Id. The pattern "TST-[0-9]+" would match (returning the same string as Issue Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:[(TST-[0-9]+)]" would match "Issue:[TST-1]" using issue Id TST-1.

In the example below, the pattern identifies the issue as belonging to a specific VersionOne project. Whenever AnthillPro comes across this regular expression, it will automatically add the appropriate comment in VersionOne.
- **Title.** Give the note title that VersionOne will use to identify this comment.
 - **Additional Comment.** Use this field to add any additional information to the VersionOne commit comment.
 - **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.

- **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
- **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
- **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
- **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.
- **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it.

5. Click **Save**.

6. **Create Defect.** Select the **Insert After** icon of the **Add Comments** step. Go to **Issue Tracking > VersionOne**, select the **Create Defect** step, and click **Select**. This step will create a defect if a failure occurs.

- **Name** the step.
- **Description.** Provide a short description.
- **Project Key.** Enter the VersionOne project key.
- **Summary.** Enter the issue summary.
- **Issue Description.** Enter the description.
- **Team.** Give the VersionOne defect team.
- **Theme.** Give the VersionOne defect theme.
- **Iteration.** give the VersionOne defect iteration.
- **Show Additional Options.** See Show Additional Options.

7. Click **Save**.

8. **Publisher Defect Report.** Select the **Insert After** icon of the **Create Defect** step. Go to **Issue Tracking > VersionOne**, select the **Publisher Defect Report** step, and click **Select**.

- **Name** the step.
- **Description.** Provide a short description.
- **Report Name.** If a new name is not given, the step name will be used.
- **Issue Id Pattern.** Give a regular expression to locate a VersionOne defect Id within changelog comments. Add "(" around the portion of the pattern which identifies the actual Issue Id. The pattern "TST-[0-9]+" would match (returning the same string as Issue Id) "TST-1" and "TST-932415", but not match "TST-a", "TST-", or "TST.1". The pattern "Issue:\\[(TST-[0-9]+)\\]" would match "Issue:[TST-1]" using issue Id TST-1.
- **Include Dependencies.** Check to include change logs from dependencies when searching for issues.
- **Show Additional Options.** See Show Additional Options.

9. Click **Save**.

Deploy Job with VersionOne Post-deploy Step

The Resolve Defect step is typically used as part of a post-deployment job. Once the artifacts have been deployed (in the example there are two artifact sets: Database and Webapp), the Resolve Defect step is run as a separate job in order to ensure that all VersionOne defects are resolved.

1. Go to **Administration**, select the appropriate **project**, and click the **Add Job** icon.
2. On the **New Job Configuration** page, choose **No** (do not use the Job Wizard). Click **Select**.
3. Follow the steps for creating a build job.
4. Select the **Insert After** icon of the step prior to the point where the VersionOne step is to be included (Artifact Delivery step in this example). Go to **Issue Tracking > VersionOne**, select the **Resolve Defect** step, and click **Select**.
 - **Name** the step.
 - **Description**. Provide a short description.
 - **Issue Key**. Enter the issue key to be resolved in VersionOne.
 - **Status**. Give the VersionOne status to set the defect to. If none is provided, it will be marked as Resolved.
 - **Resolution Reason**. Provide a reason for resolving a defect.
 - **Resolution Details**. Input additional information regarding the resolution of this defect.
 - **Close Issue**. Check the box to automatically mark the issue as closed. If the box is not checked, the issue will be marked as resolved.
 - **Show Additional Options**. See Show Additional Options.
 - Click **Save**.

Add VersionOne Jobs to Workflows

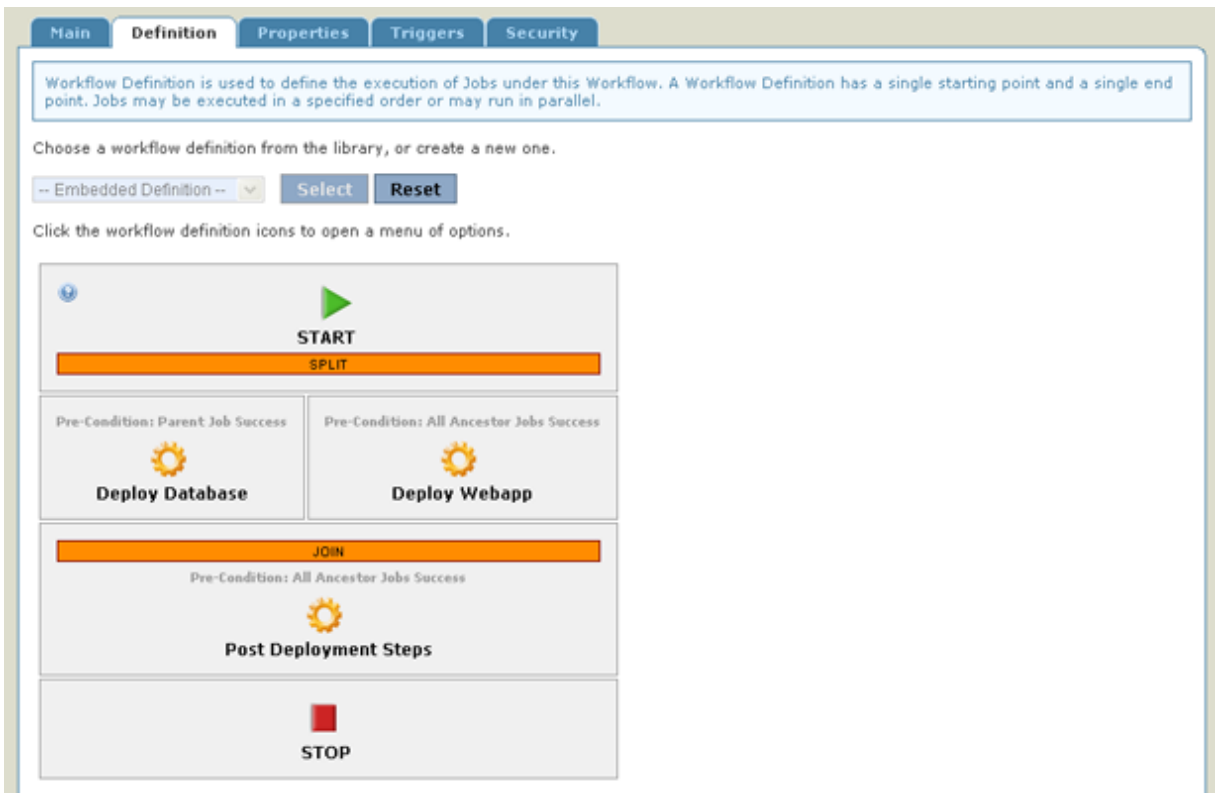
The Jobs created in the Configure Jobs section must be executed as part of a workflow. This section will assume that an originating workflow has already been configured, and will cover the process of adding the VersionOne jobs to the appropriate workflows. Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the VersionOne integration.

Build Workflow with VersionOne Steps

1. Go to **Administration**, select the **project**, and select the **build** workflow.
2. On the workflow **Main** page, select the **Definition** tab.
3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
4. Left-click the **Start** icon and select **Insert Job After**.
5. Select the **Build** job created in the Configure Jobs section, a **job pre-condition** script, and click **Insert Job**.

Deploy Workflow with VersionOne Post-deploy Step

1. Go to **Administration**, select the **project**, and select the **deploy** workflow.
 2. On the workflow **Main** page, select the **Definition** tab.
 3. From the drop-down menu, choose **Embedded Definition** and click **Select**.
 4. Left-click the **Start** icon and select **Insert Job After**.
 5. Select the **Post Deployment Steps** job created in the Configure Jobs section, a **job pre-condition** script, and click **Insert Job**.
- Note that this jobs deploys two artifact sets and that the Post Deploy Steps job will change the VersionOne issue status for both artifact sets.



Run Workflows and View Reports (VersionOne)

1. Go to **Dashboard** and select the **workflow** created in the Create Workflow section.
2. On the workflow **Main** page, click the **Build** button for the workflow.
 - This procedure is the same for the build and deploy workflows.
3. Once the **workflow** has completed, select the appropriate **Build Life** and click the **Reports** tab.

Version One DEFECT REPORT

Project: [Maven-Example \(VersionOne\)](#)
 BuildLife: [285](#)
 Generated On: 03/28/2008 11:31 AM -0400
 Latest Stamp: 22
 Version One Server: <http://versionone/VersionOne/>

Defect ID: D-01007

Issue Type: Defect
 Name: My name is Nathaniel!
 Description: I like to log defects!
 Status: In Progress

4. Select a **link** to view a report. To go to the issue that was generated in VersionOne, follow the **Issue Id** link.
5. Click the **Issues** tab to view the issues AnthillPro created in VersionOne.



6. To drill down on each VersionOne step on the Build Life Summary page, see Trace a Build Life to Source.

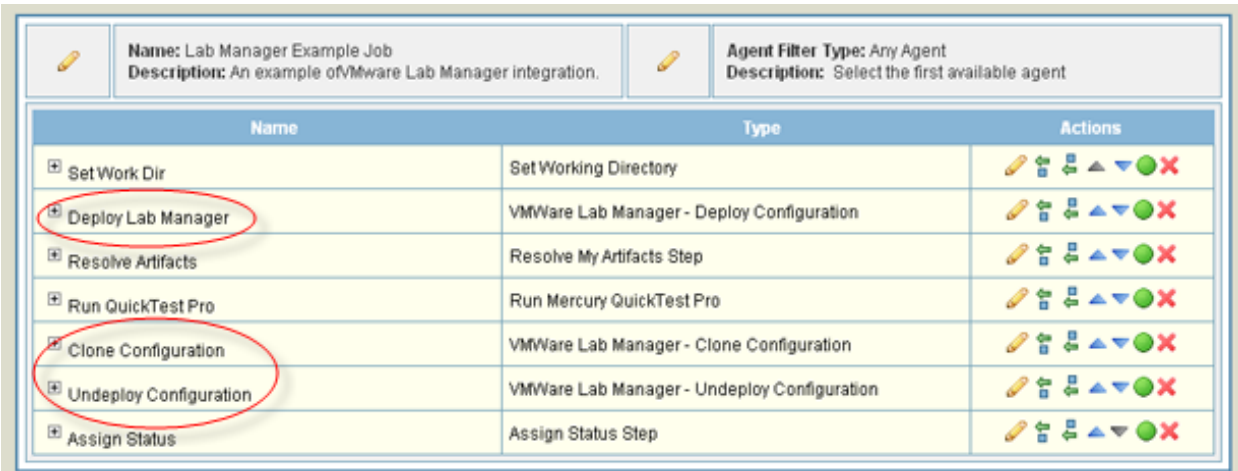
Chapter 69. Virtualization Tools

The Virtualization integration allows AnthillPro to clone, deploy, and undeploy configurations. Implemented as job steps, the integration can be used as part of your build workflows or as a secondary process.

VMware Lab Manager

The VMware Lab Manager (2.5.x and above) integration provides control environments used by AnthillPro, as well as those not directly related to AnthillPro. The integration allows AnthillPro users to ensure that specific sets of machines (e.g., DEV, QA, PROD, etc.) are available for use by the AnthillPro server; as well as the use of Lab Manager load management capabilities to maximize server resources.

The Lab Manager integration is implemented as AnthillPro job steps, and is configured on the Job Configuration page. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab.



Name	Type	Actions
Set Work Dir	Set Working Directory	[Pencil] [Add] [Remove] [Up] [Down] [Stop] [Refresh]
Deploy Lab Manager	VMWare Lab Manager - Deploy Configuration	[Pencil] [Add] [Remove] [Up] [Down] [Stop] [Refresh]
Resolve Artifacts	Resolve My Artifacts Step	[Pencil] [Add] [Remove] [Up] [Down] [Stop] [Refresh]
Run QuickTest Pro	Run Mercury QuickTest Pro	[Pencil] [Add] [Remove] [Up] [Down] [Stop] [Refresh]
Clone Configuration	VMWare Lab Manager - Clone Configuration	[Pencil] [Add] [Remove] [Up] [Down] [Stop] [Refresh]
Undeploy Configuration	VMWare Lab Manager - Undeploy Configuration	[Pencil] [Add] [Remove] [Up] [Down] [Stop] [Refresh]
Assign Status	Assign Status Step	[Pencil] [Add] [Remove] [Up] [Down] [Stop] [Refresh]

When configuring the Lab Manager integration, it is possible to include scripts and/or property lookups in all the fields. For example, specify `#{bsh: . . . }` or `#{property: . . . }` type strings that will be resolved to values at runtime (see Scripting). Any steps within AnthillPro relying on VMware Lab Manager will not work until the integration is configured. The Lab Manager integration includes the following AnthillPro job steps (in the VMware folder):

- **Lab Manager - Deploy Configuration.** Deploys a Lab Manager configuration. This step will locate a configuration by name and start it if the configuration is deployed. If the configuration is not deployed, this step will deploy the configuration first.
- **Lab Manager - Undeploy Configuration.** Undeploys a Lab Manager configuration. This step will locate the configuration by name and undeploy the configuration. If the configuration is undeployed the step will do nothing.
- **Lab Manager - Clone Configuration.** Clones a Lab Manager configuration. This step will create a clone of any specified configuration and then identify the clone with a user specified name.

Lab Manager Prerequisites

- You must have AnthillPro administrative privileges to configure the integration. See Manage Security.

- VMware Lab Manager 2.5.x or greater.
- Lab Manager base URL, user name, and password.
- Necessary permissions set in Lab Manager that allow the specified AnthillPro user to access the web-service interface. *If this is not done, Lab Manager may display an alert saying only the administrator has permission to use the web service.*

Configure Lab Manager

1. Go to **System > VMware Lab Manager** under the Integration menu.
2. On the **VMware Lab Manager Integration** page, click **Edit**.
3. Configure:
 - Give the **Lab Manager Base URL**.
 - **User Name**. Enter the user name to be used to connect to the Lab Manager server. This name needs to be pre-configured in Lab Manager and is used for authentication, allowing AnthillPro to perform Lab Manager tasks.
 - **Password**. Provide the password to be used to connect to the Lab Manager.
 - **Password Script**. To use a script or property lookups for the password, leave the Password field blank and enter it here. The password script is used to retrieve the password from external source. Essentially, this is a BeanShell script that reads the password from a file (or contacts an external service to retrieve the password) and populates the password field at runtime. This feature is most useful when passwords are changed often. See Scripting.
 - **HTTP Connection Timeout**. Enter the time (in minutes) it takes for the slowest Lab Manager configuration to deploy to a server.
4. Click **Set** then **Done**.

Adding Lab Manager Steps to an Existing Job

Any Lab Manager step can be used as part of an existing job. Where the step is placed within the job depends on what the job does. The example in this section will copy and save a Lab Manager configuration to be reproduced later. When using the integration, click the Create Step button (or select the Insert After/Before icon) to add steps to a job. Once the job is configured, it is then added to the workflow under the Definition tab.

Though the example embeds a single job within a workflow, it is possible to use the Lab Manager integration in a number of ways. The integration can be used with originating or non-originating workflows, and can be run sequentially or in parallel with other jobs within the same workflow. For simplicity, the Lab Manager steps have been included as part of a single build job in the example.

1. Go to **Administration**, select the appropriate **project**, and then select the **job** that the Lab Manager steps are to be added to.
2. **Lab Manager - Deploy Configuration**. Select the **Insert After** icon of the step prior to where the Deploy Configuration step is to be included (the Set Working Directory step for example). Expand the **VMware** folder, select the **Lab Manager - Deploy Configuration** step, and click **Select**.
 - **Name** the step.

- **Description.** Provide an optional description.
- **Configuration Name.** Give the Lab Manager configuration name (a script may be used).

If using the **Wait for Agents** option below, the configuration name must be an exact match to the property set on the agent, or the deploy configuration step will fail. Any punctuation, capitalization, and/or spaces must be identical. For example, if the configuration name is test.config, the property on the agent must be named lab.manager.test.config (not lab.manager.test-config, lab.manager.test config, etc.). If a script is used to determine the configuration name, it must return a name that exactly matches the agent property set below.

- **Fence Mode.** Select the Lab Manager options to fence; to block traffic in and out; to allow traffic out only; or to allow traffic in and out.
- **Wait for Agents.** Select this option to have AnthillPro wait for agents within the configuration to come online. The default timeout value for this step is set to 10 minutes to prevent unsuccessful deploys from blocking the workflow execution. If your deploys run longer, you may need to adjust this setting (see Show Additional Options below).

For an agent to be considered a member of a configuration, it must contain an agent property of the form lab.manager.<configurationName>, where configurationName is the name of the configuration being deployed. For example, if the configuration name is test.config, the property set on the agent must be lab.manager.test.config (only the agent property name is used, so there are no requirements for the property value). Once the agents in the configuration have this property, AnthillPro will wait until the number of agents defined below are online.

When setting the agent property, first make sure the agent is installed (see Installing AnthillPro) within the configuration to be deployed and has been added to the correct AnthillPro environment(s) (see also Configure Agents). With the agent online, add the property. See here in the Configure Agents section.

- **Number of Agents to Wait For.** If the Wait for Agents option is selected, specify the minimum number of agents to come online before the step completes. With the default value, 0 (zero), AnthillPro will wait for 1 agent from each machine in the configuration (a value of 1 is equivalent).

While waiting for agents to come online, AnthillPro will look for the agent property set above, so the number of agents set here may not exceed the number of agents that contain the property set in the previous step -- otherwise the step will timeout. For example, if the lab.manager.test.config property is set on 3 agents within the configuration test.config, and the number of agents to wait for is set at 4, the step will fail. However, if the lab.manager.test.config property is set on 3 agents within the configuration test.config, and the number of agents to wait for is set at 1, the step will complete (i.e., AnthillPro will not wait for all 3 agents to come online before completing the deploy configuration step).

The default timeout value for this step is set to 10 minutes to prevent unsuccessful deploys from blocking the workflow execution. If your deploys run longer, you may need to adjust this setting (see Show Additional Options below).

- **Show Additional Options (advanced).** Select the Show Additional Options link to configure more options.
 - **Is Active.** Select No to temporarily deactivate the step without deleting it; otherwise select Yes.
 - **Pre-Condition Script.** From the drop down menu, select the condition which must be met for the step to continue. Before editing an existing script or creating a new one, see Step Pre-Condition Scripts.
 - **Ignore Failures.** Select Yes if this step should not effect the determination for step continuation or the status determination of the job.
 - **PostProcessingScript.** Select a script for determining when commands should count as fail or succeed. See Post Processing Scripts.

- **Timeout.** Enter the time in minutes after the start of the step when AnthillPro will consider the step as timed out and abort it. The default wait time is set to 10 minutes. If it takes longer to deploy, increase this default value. This value will be used to determine how long AnthillPro will wait for the agents specified above.
3. Click **Save**.
 4. **Lab Manager - Clone Configuration.** Select the **Insert After** icon of the step prior to where the Clone Configuration step is to be included (the Run QTP Test step for example). Expand the **VMware** folder, select the **Lab Manager - Clone Configuration** step, and click **Select**.
 - **Name** the step.
 - **Description.** Give a description.
 - **Configuration Name.** Provide the name of the Lab Manager configuration to be closed.
 - **Cloned Configuration Name.** Give a name for the cloned configuration. This should be different from the original.
 - **Show Additional Options (advanced).** See Show Additional Options.
 5. Click **Save**.
 6. **Lab Manager - Undeploy Configuration.** Select the **Insert After** icon of the Lab Manager - Clone Configuration step, expand the **VMware** folder, select **Lab Manager - Undeploy Configuration**, and click **Select**.
 - **Name** the step.
 - **Description.** Give a description.
 - **Configuration Name.** Provide the name of the Lab Manager configuration to be closed.
 - **Show Additional Options (advanced).** See Show Additional Options.
 7. Click **Save**.

Adding Lab Manager Job to Workflow

This example adds a job to an existing workflow. Complete workflow configuration is beyond the scope of this tutorial. The topics covered in detail below are specific to using the Lab Manager integration.

1. Go to **Administration**, select the **workflow** the Lab Manager job is to be added to.
2. Select the **Definition** tab, choose **Embedded Definition** from the drop-down menu, and click **Select**.
3. Left-click the **START** icon and select **Insert Job After**.
4. Select the **Job** from the drop-down menu, choose a **pro-condition script**, and click **Insert Job**.

Chapter 70. Using AnthillPro Plugins

With the AnthillPro Plugin system, you can write your own integration with third-party tools (e.g., home-grown, testing, SCM, source-code analytic, etc.) and then add them to your AnthillPro processes. In addition, most new integrations written by Urbancode will be Plugins and included when you install a new server (see Activating and Deactivating Plugins).

It is possible to add a Plugin without having to upgrade the server. For example, if Urbancode writes a new Plugin, you can download it from Supportal [<http://support.urbancode.com/>] and then follow the steps below to add it to your AnthillPro server.

Once a Plugin has been written (to write your own Plugin, see Developing Plugins), upload it to the AnthillPro server on the System page. A copy of the zipped Plugin will then be stored in the server's /plugin directory. Upon successful upload, configure most Plugins on the System page (e.g., if you are using an SCM Plugin, it will need to be configured by going to the Repositories page; or if the Plugin is a testing tool, configure it on the Integration page, etc.). Then, add the integration (Plugin) to your projects (see Adding a Plugin to a Project).

To begin using an existing Plugin, see Uploading a Plugin.

Plugin Prerequisites

- You must have permissions to the System page to upload and configure the Plugin.
- To add the Plugin to existing projects, you will need permissions to the Administration page.

Uploading a Plugin

Once a Plugin has been written (to write your own Plugin, see Developing Plugins), upload it to the AnthillPro server on the System page. A copy of the zipped Plugin will then be stored in the server's /plugin directory.

1. Go to **System > Plugins** under the Server menu.
2. **Load a Plugin.** Select the Browse button and upload the Plugin. The Plugin must be contained in a .zip file. Click **Load**.
3. When the upload is complete, you can view the details of the Plugin by either selecting the magnifying glass under the Operations menu or selecting the Plugin's name.
4. To delete an existing Plugin, click the Delete icon under the Operations menu. If you get a warning when attempting to delete a Plugin, that means it is in use by at least one project. To delete a Plugin that is in use, you must remove it from all existing job configurations.
5. Click **Done**. See Configuring a Plugin.

Configuring a Plugin

Once the Plugin has been successfully uploaded (see Uploading a Plugin), you will need to configure it on the System page before it is available to your AnthillPro projects (see Adding a Plugin to a Project). Typically, once the Plugin upload is complete, it will appear under either the Integration or Repositories menu on the System page. To

configure the Plugin integration, select it from the menu and then give the necessary information.

If you are familiar with existing AnthillPro integrations, most of the information required to complete the configuration will be similar. However, because each Plugin is different, complete configuration instructions are not possible. If you are having trouble configuring the Plugin, please contact the author for assistance.

Once the configuration is complete, see [Adding a Plugin to a Project](#) to begin using the new integration.

Adding a Plugin to a Project

Typically, a Plugin is added to a project as a job step, similar to traditional AnthillPro job configuration. If you are moving from a current integration to a Plugin version, you may need to reconfigure jobs.

Activating and Deactivating Plugins

AnthillPro ships with pre-configured integrations that are written as Plugins. During installation, the user was asked which Plugins (i.e., integrations) to deactivate. If you want to use a deactivated Plugin, you will need to activate it first. Assuming you have administrative permissions, simply click the brown icon.

You can deactivate a Plugin by clicking on the green active icon. Once you confirm that you want to deactivate it, the Plugin will not be available. If the Plugin is in use by any project, AnthillPro will notify you. If you elect to deactivate a Plugin that is in use, the steps that use the Plugin will be disabled.

If you reactivate a Plugin that is in use, you will need to manually enable any job steps that use the Plugin.

Part XIII. Security

In AnthillPro, you have detailed control over what a user can see and do. The system enables you to map your organizational structure by teams, activities, etc. For example, you can set up AnthillPro so that a dev team only see the projects they work on, or the QA team can only access the build artifacts. Security management begins with Roles. In turn, each Role has corresponding Permissions to either restrict or allow a user to perform tasks, view pages, etc. Once the Roles and Permissions have been configured, Authorization Realms realms and then Authentication Realms are configured. Once security is configured, Audits may be performed for the who-when changes Administrative users make to the system.

Chapter 71. Setting Up Security

When setting up security, you will need to follow a specific series of steps to configure the security system. The steps are as follows:

1. **Roles.** If not using the default Roles, create new roles. See Define Roles.
2. **Permissions.** Determine which Roles have what permissions by Resource and Resource Type, if you are not using the Default Permissions. See Set and Manage Permissions.
3. **Authorization Realms.** If not using the default Authorization Realms, one must be created. See Configure Authorization Realms.
4. **Authentication Realms.** If not using the default Authentication Realms, one must be created. See Configure Authentication Realms.
5. **Users.** Add users (which are associated with Roles) to the appropriate Authentication Realm. See Add Users. *If using LDAP, see also Configure Authorization Realms.*

In addition, if you use LDAP or other similar tools, you can import users into AnthillPro, and then map them to the security system. To do this, you'll need the appropriate permissions to that tool that allow AnthillPro to access it.

Define Roles

AnthillPro uses a flexible, role-based security model. Roles created and edited in this section as the building block to create security schemes. The default Roles that ship with AnthillPro can be edited to meet specific needs. If additional Roles are necessary, new ones may be created. You will need access to the System page in order to manage security.

1. Go to **System > Roles** under the Security menu.
2. On the **Roles** tab, click the **Create Role** button.
3. Configure Role:
 - **Name** the Role.
 - **Require Secondary Authentication.** Check the box to require Secondary Authentication.

Once a primary Authentication Realm is configured, the Secondary Authentication Realm adds a second layer of security. *If this item is checked, every Authentication Realm that this Role belongs to will require that Secondary Authentication be configured.* See Configure Authentication Realms.

4. Click **Save** then **Done**.

Using Permissions

Permissions associate the role, resource, and an action that may be performed on the resource. Typical actions include the ability to read or view the resource (i.e., a project, workflow, agent, etc.); the ability to write to or modify the resource (e.g., add an agent property); the ability to modify the security settings for a resource; or the ability to execute (in the case of workflows) the resource. Generally, permissions fall within one of these groups:

- **Read or View**, depending on the resource. Users assigned this permission can read (view), a resource, but will not be able to create or change a configuration. For example, a user with "read" permissions to an agent will be able to see that agent within the user interface, but will not have access to configure that agent.
- **Write** permission includes the "read" permission, with the additional ability to perform a task using a specific resource. For example, a user with "write" permissions to an originating workflow can modify the workflow.
- **Execute (workflow only)**. Allows a user to run a particular workflow. Anyone with access to execute a build workflow must have at least read permissions. If the user must run a secondary workflow in a different environment, that user must also be assigned "use" permissions to the target environment(s).
- **Task Execute (workflow only)**. Allows the user to perform the same actions as the execute permission, but only through the Task interface. Note that the "task execute" permission does not allow users to manually run a secondary process, etc., and gives them no system permissions. This is useful in situations where AnthillPro users are only tasked with deploying, but not building, a project or workflow.
- **Security** permission allows users to change the security settings for a specific resource. For example, a user with "security" permissions to an agent can determine which users can view, configure, and set security for that agent.

By modifying the Default Permissions and/or manually configuring additional permissions, you can fine tune what users are able to access which resource. Permissions are available for the following resource types (for workflows, see Setting Workflow Permissions):

- **General Resources.** Users may be assigned **Read**, **Write**, **Use**, and/or **Security** permissions for the most commonly accessed resources:

Agent	Distributed Server	Library Job (Configuration)	Project
Agent Relay	Environment	Library Workflow Definition	Repository
Artifact Set (see Securing Artifact Sets)	Environment Group	Life-Cycle Model	Script Group
Codestation Project	Folder		

- **System Resources.** AnthillPro provides security around System resources that only select groups of users need access to. The following resources typically require **Read** and **Security** permissions for administrators in order to configure/edit settings:

Integration	Project Support	Script Library	Server
Notification	Reporting	Security	

- **System Functions.** A user must have system permissions to act as administrator of the system-level functions listed below. **Read** or **Security** permissions provides a user access to the System page. A user may be assigned permissions to the following System Function resources:

Administration	Integration Administration	Report Administration	Script Administration
Agent Administration	Life-Cycle Model Administration	Reporting	Security Administration
Auditing	Notification Administration	Repository Administration	Server Administration
Delete Runtime History	Prioritize Workflows	Restart Workflows	Stamp Administration
Environment Administration	Project Administration	Schedule Administration	System Administration

Configure Default Permissions

When a new resource is created, it is automatically assigned Default Permissions by the AnthillPro system. You can modify these default permissions by following the procedures outlined in the Assign Additional Permissions section (i.e., editing the Default Permissions section of the resource). As a rule, the Default Permissions should allow most users within a role to perform their work. If you are frequently assigning additional permissions to a role, consider changing the default permissions for newly created resources.

- Please note that changes made to the default permissions will only effect new, and not existing, settings.

Assign Additional Permissions

When a new resource type is created, it is automatically assigned the predefined Default Permissions (see Configure Default Permissions). For most users, the Default Permissions should be adequate. But there are circumstances in which a small subset of users will need different/additional permissions to a resource. If this is the case, manually setting permissions by resource type is necessary. For example, the default permissions for the "User" role does not allow people to perform "write" tasks for a Codestation project. However, these users must occasionally update a third-party library. In this case, you can simply select Codestation Project from the Resource Type menu, find the project in the list, and add the write permission to the appropriate role.

1. Go to **System > Permissions** under the Security menu.
2. Select a **Resource Type** from the drop-down menu and click **Set**.
3. Select the **Role** from the Add Permission drop-down menu to add the read, write, and/or security actions that role can perform on the resource.

If you are changing the Default Permissions, please see Configure Default Permissions for more.

4. To remove a role from performing an action click the **delete** icon under the Roles menu.
5. Click **Add** the **Done**.

Setting Workflow Permissions

Workflows have their own permissions, different from other resources, but are configured like the other permission types (see Assign Additional Permissions). Users tasked with running either an originating, operational, and/or secondary workflow must have at least execute permissions to the workflow(s). In addition, that user must have access to the environment(s) and/or agent(s) used by they project(s) they execute workflow for.

Workflows have the following permissions you can assign to them:

- **Execute.** Allows a user to run a particular workflow. Anyone with access to execute a build workflow must have at least read permissions. If the user must run a secondary workflow in a different environment, that user must also be assigned at least "use" permissions to the target environment(s).
- **Task Execute.** Allows the user to perform the same actions as the execute permission, but only through the Task interface. Note that the "task execute" permission does not allow users to manually run a secondary process, etc., and gives them no system permissions. This is useful in situations where AnthillPro users are only tasked with deploying, but not building, a project or workflow.
- **Security.** Allows a user to change the resource's security settings.

For most users, the Default Permissions (see Configure Default Permissions) should be adequate. But there are circumstances when a small subset of users will need different permissions to a workflow. In this case, you can manually assign additional permissions. For example, the default permissions for the "User" role do not allow people to execute a deployment (secondary) workflow. However, you have one project in which you want these people to run a deployment using the Task interface. In this case, you can simply select workflow from the Resource Type menu, find the deployment workflow in the list, and add the Task Execute permission to the appropriate role.

Configure Authorization Realms

Authorization Realms are used by Authentication Realms to associate Users with Roles and to determine user access to AnthillPro. Authorization Realms can be deactivated by clicking the **Mark as Inactive** icon (see below).

There are three Authorization Realm options:

- **Anthill.** Uses internal Anthill role management. See Anthill Authorization Realm.
- **LDAP.** Uses external LDAP role management. See LDAP Authorization Realm.
- **Single Sign-On.** Uses external Single Sign-On role management. See Single Sign-On Authorization Realm.

Anthill Authorization Realm

The Anthill Authorization Realm uses AnthillPro to manage Users. You will need access to the System page in order to manage security.

1. Go to **System > Authorization** under the Security menu.
2. On the **Authorization** tab, click the **Create Authorization Realm** button.
3. Check **Anthill**, click **Set**, and configure Realm.
 - **Name** the Authorization Realm.
 - **Description.** Provide a description of the Authorization Realm.
4. If not adding an initial User Role, Click **Save** then **Done** to complete. Otherwise proceed to **item 5**.
5. Click the **Add Initial User Role** button (optional).

New users created within an Authentication Realm governed by the Authorization Realm you are configuring will automatically become members of the Roles configured here. For example, if a user is added to the Kerberos Authentication Realm that is managed by the Anthill Authorization Realm, the new user will be automatically assigned the Roles chosen here.

This item requires that the appropriate Roles have already been created. See Define Roles.

6. Select the Role from the drop-down menu and click **Add Role**.
7. To add more User Roles, repeat items 5 and 6.
8. Click **Save** then **Done**.

LDAP Authorization Realm

The LDAP Authorization Realm uses an external LDAP server for authorization. If User Roles are defined in LDAP as an attribute of the User, the LDAP Role Attribute configuration must be used. If User Roles are defined elsewhere in LDAP *and* reference the Users that belong to them, a LDAP Role Search needs to be performed. You will need access to the System page in order to manage security.

1. Go to **System > Authorization** under the Security menu.
2. On the **Authorization** tab, click the **Create Authorization Realm** button.
3. Check **LDAP** and click **Set**.
4. On the **Main** tab, configure Realm:
 - **Name** the Authorization Realm.
 - **Description**. Provide a description of the Authorization Realm.
 - **Role Attribute**. Give the name of the attribute that contains role names in the user directory entry.
*If User Roles are defined in LDAP as an attribute of the User, the **Role Attribute** configuration must be used.*
 - **Role Name**. Provide the name of the entry that contains the user's role names in the directory entries returned by the role search. If this is not specified, no role search will take place.
*If User Roles are defined elsewhere in LDAP **and** reference the Users that belong to them, a Role Search needs to be performed.*
 - **Role Base**. Give the base directory to execute role searches in (e.g., ou=groups,dc=anthill3,dc=com).
 - **Role Search**. Provide the LDAP filter expression to use when searching for user role entries. The user name will be put in place of {1} in the search pattern and the full user DN will be put in place of {0} (e.g., member={0}).
 - **Search Role Subtree**. Check **True** to search the subtree for the roles or **False** to not search.
5. If not mapping **LDAP roles** to Anthill Security Roles, click **Save** then **Done** to complete. Otherwise proceed to **item 6**.
6. Select the **Role Mapping** tab (optional) and follow the **Map LDAP Role** link.
This item requires that the appropriate Roles have already been created. See Define Roles.
 - **LDAP Role Name**. Give the **LDAP** role to map.
 - **Anthill Role**. Select the **Anthill** role to map the LDAP role to.
7. Click **Save** then **Done**.

Single Sign-On Authorization Realm

The Single Sign-On Authorization Realm uses an external Single Sign-On server for authorization. You will need access to the System page in order to manage security.

For authorization to complete, you will need to add the AnthillPro ROOT URL to your Single Sign-On authorization server.

1. Go to **System > Authorization** under the Security menu.
2. On the **Authorization** tab, click the **Create Authorization Realm** button.
3. Check **Single Sign-On** and click **Set**.
4. On the **Main** tab, configure Realm:
 - **Name** the Authorization Realm.
 - **Description**. Provide a description of the Authorization Realm.
 - **Roles Header Name**. Enter the name of the HTTP header that contains a comma delimited list of Single Sign-On roles.
5. Click **Save**.
6. If not mapping **Single Sign-On roles** to Anthill Security Roles, click **Done** to complete. Otherwise proceed to **item 7**.
7. Select the **Role Mapping** tab and follow the **Map Single Sign-On Role** link.

This item requires that the appropriate AnthillPro Roles have already been created. See Define Roles.

 - Give the **Single Sign-On** role to map.
 - Select the **AnthillPro** role to map the Single Sign-On role to.
8. Click **Save** then **Done**.

Configure Authentication Realms

Create and edit Authentication Realms to determine a users identity within an Authorization Realm. The User authentication is determined following the hierarchy of realms displayed on the Authentication tab. In the example below, authentication will first be determined in the System Realm, followed by LDAP and Anthill Realms, so a user listed in the LDAP realm may have different authorizations from those in the other realms.

If you have a number of authentication realms, you can reorder them using the drag-and-drop tool so that the ones you are most interested in appear at the top of the list. Realms (except for the default Anthill realm) can also be **activated or deactivated** using the **Operations** menu.

There are six Authentication Realm options:

- **Anthill**. Uses AnthillPro to manage user authentication. See Anthill Authentication Realm.
- **Custom**. Use this option to create your own authentication LoginModule. See Custom Authentication Realm.
- **Kerberos**. External Kerberos user authentication using GSS-API. See Kerberos Authentication Realm.
- **LDAP**. External LDAP integrated user authentication. See LDAP Authentication Realm.
- **RSA SecurID**. External Single Sign-On user authentication. See RSA SecurID Authentication Realm.
- **Single Sign-On**. External Single Sign-On user authentication. See Single Sign-On Authentication Realm.

You can add a second layer of security on the **Secondary Authentication** tab. Upon authentication in the primary Realms, Users are forwarded to this secondary Realm if they are a member of any Role requiring secondary authentication. See the Secondary Authentication item for the Authentication Realms you are using.

Anthill Authentication Realm

The Anthill Authentication Realm uses AnthillPro to internally manage users. You will need access to the System page in order to manage security.

1. Go to **System > Authentication** under the Security menu.
2. On the **Authentication** tab, click the **Create Authentication Realm** button.
3. Check **Anthill** and click **Set**.
4. On the **Authentication** tab, configure Realm:
 - **Name** the Authorization Realm.
 - **Description.** Provide a description of the Authorization Realm.
 - **Authorization Realm.** Select the Authorization Realm this Authentication Realm will use.

This item requires that the appropriate Authorization Realm has already been created. See Configure Authorization Realms.
5. If not setting a **Secondary Authentication Realm**, click **Save** then **Done** to complete. Otherwise proceed to **item 6**.
6. Select the **Secondary Authentication** tab (optional) and click the **Create Secondary Authentication Realm** button.

Upon authentication in the primary Realms, Users are forwarded to this secondary Realm if they are a member of any Roles which require it. The Secondary Authentication Realm adds a second layer of security.

This item requires that the appropriate Roles and Authorization Realms have already been created. See Define Roles and Configure Authorization Realms.

7. Check one of the available Authentication Realms and click **Set**.
8. Configure Secondary Authentication.

Authentication Settings will only take effect if the test login is successful (if test fails, a warning will appear).

 - **Name** the Secondary Authentication Realm.
 - **Description.** Provide a description of the Secondary Authentication Realm.
 - **Test User Name.** Give the user name to test the configuration with.
 - **Test Passcode.** Provide the passcode to test the configuration with.
9. Click **Save** then **Done**.

Custom Authentication Realm

The Custom Authentication Realm uses a specified Java LoginModule to authenticate users. You will need access to the System page in order to manage security.

1. Go to **System > Authentication** under the Security menu.
2. On the **Authentication** tab, click the **Create Authentication Realm** button.
3. Check **Custom**, click **Set**.
4. On the **Authentication** tab, configure Realm:
 - **Name** the Authorization Realm.
 - **Description.** Provide a description of the Authorization Realm.
 - **LoginModule Class.** Give the class name of the Java LoginModule class to use (e.g., com.mycompany.security.loginmodule).
 - **Authorization Realm.** Select the Authorization Realm this Authentication Realm will use.
This item requires that the appropriate Authorization Realm has already been created. See Configure Authorization Realms.
5. If not setting a **Secondary Authentication Realm**, click **Save** then **Done** to complete. Otherwise proceed to **item 6**.
6. Select the **Secondary Authentication** tab (optional) and click the **Create Secondary Authentication Realm** button.

Upon authentication in the primary Realms, Users are forwarded to this secondary Realm if they are a member of any Roles which require it. The Secondary Authentication Realm adds a second layer of security.

This item requires that the appropriate Roles and Authorization Realms have already been created. See Define Roles and Configure Authorization Realms.
7. Check one of the available Authentication Realms and click **Set**.
8. Configure Secondary Authentication. *Authentication Settings will only take effect if the test login is successful (if test fails, a warning will appear).*
 - **Name** the Secondary Authentication Realm.
 - **Description.** Provide a description of the Secondary Authentication Realm.
 - **Test User Name.** Give the user name to test the configuration with.
 - **Test Passcode.** Provide the passcode to test the configuration with.
9. Click **Save** then **Done**.

Kerberos Authentication Realm

The Kerberos Authentication Realm uses Kerberos to authenticate users. You will need access to the System page in order to manage security.

1. Go to **System > Authentication** under the Security menu.

2. On the **Authentication** tab, click the **Create Authentication Realm** button.

3. Check **Kerberos** and click **Set**.

4. On the **Authentication** tab, configure Realm:

- **Name** the Authorization Realm.
- **Description.** Provide a description of the Authorization Realm.
- **Conf File.** Give the path to the Kerberos configuration file. On Unix and Linux, it is usually */etc/krb5.conf*.
- **Authorization Realm.** Select the Authorization Realm this Authentication Realm will use.

This item requires that the appropriate Authorization Realm has already been created. See Configure Authorization Realms.

5. If not setting a **Secondary Authentication Realm**, click **Save** then **Done** to complete. Otherwise proceed to **item 6**.

6. Select the **Secondary Authentication** tab (optional) and click the **Create Secondary Authentication Realm** button. The Secondary Authentication Realm adds a second layer of security.

Upon authentication in the primary Realms, Users are forwarded to this secondary Realm if they are a member of any Roles which require it.

This item requires that the appropriate Roles and Authorization Realms have already been created. See Define Roles and Configure Authorization Realms.

7. Check one of the available Authentication Realms and click **Set**.

8. Configure Secondary Authentication. *Authentication Settings will only take effect if the test login is successful (if test fails, a warning will appear).*

- **Name** the Secondary Authentication Realm.
- **Description.** Provide a description of the Secondary Authentication Realm.
- **Test User Name.** Give the user name to test the configuration with.
- **Test Passcode.** Provide the passcode to test the configuration with.

9. Click **Save** then **Done**.

LDAP Authentication Realm

The LDAP Authentication Realm uses an external LDAP server for authentication. You will need access to the System page in order to manage security.

If you plan on using SSL, please see Using LDAP over SSL with Self-signed Certificate below before continuing.

1. Go to **System > Authentication** under the Security menu.

2. On the **Authentication** tab, click the **Create Authentication Realm** button.

3. Check **LDAP** and click **Set**.

4. On the **Authentication** page, configure the following:

- **Name** the Authorization Realm.
- **Description.** Provide a description of the Authorization Realm.
- **Authorization Realm.** Select the Authorization Realm this Authentication Realm will use.

This item requires that the appropriate Authorization Realm has already been created. See Configure Authorization Realms.

- **Context Factory.** Give the context factory class used to connect. *This may vary depending upon your specific Java implementation.* The default for Sun Java implementations: `com.sun.jndi.ldap.LdapCtxFactory`.
- **LDAP URL.** Provide the full URL to the LDAP server, beginning with **ldap://** (e.g., `ldap://ldap.mydomain.com:389`). If using SSL, please see Using LDAP over SSL with Self-signed Certificate.

5. **If your LDAP users exist in a single directory**, select the button and give the following (otherwise, go to the next step):

- **User Pattern.** Give the user directory entry pattern. The user name will be put in place of {0} in the pattern (e.g., `cn={0},ou=employees,dc=anthill3,dc=com`).

6. **If your LDAP Users exist in many directories** AnthillPro will have to perform a User search with the user-name. To enable this, give the following:

- **User Base.** Give the user base directory to search for users in (e.g., `ou=employees,dc=anthill3,dc=com`).
- **User Search.** Provide the LDAP filter expression to use when searching for a user's directory entry. The user name will be put in place of {0} in the search pattern (e.g., `uid={0}`).
- **Search User Subtree.** If the LDAP User Names are case sensitive, check **True** to have AnthillPro treat different-case User Names as different Users or **False** to ignore.
- **Check the box if no anonymous access is allowed.** If you are unable to access the LDAP server anonymously, you can use the Connection Name and Connection Password. Give the following (as above):
 - **Connection Name.** Give the directory name to use when binding to the LDAP for searches (e.g., `cn=Manager,dc=anthill3,dc=com`). If not specified, an anonymous connection will be made. *Connection Name is required if the LDAP server cannot be anonymously accessed.*
 - **Connection Password.** Provide the password to use when binding to the LDAP for searches. Required if Connection Name is specified. *Connection Password is required if the LDAP server cannot be anonymously accessed.*

7. Complete basic configuration:

- **Case Sensitive User Names.** If you have case-sensitive user names (e.g., the user Bob is different than the user bob), select true. Otherwise, select false.
- **First Name.** To read information from the LDAP entry, give the name of the user's LDAP entry attribute that contains the first name.
- **Last Name.** To read information from the LDAP entry, provide the name of the user's LDAP entry attribute that contains the last name.
- **Email.** To read information from the LDAP entry, give the name of the user's LDAP entry attribute that contains the email address.

- **XMPP ID.** To read information from the LDAP entry, give the name of the user's LDAP entry attribute that contains the XMPP / Jabber IM ID.
 - **MSN ID.** To read information from the LDAP entry, give the name of the user's LDAP entry attribute that contains the MSN IM ID.
8. If not setting **User Filters**, click **Save** then **Done** to complete. Otherwise configure the **User Filters** to allow or block LDAP user from access to AnthillPro:
- **Allowed Patterns.** Enter line-separated regular expressions of LDAP user names that will explicitly be allowed to log in to Anthill. Leave empty to not allow all users.
 - **Blocked Patterns.** Enter line-separated regular expressions of LDAP user names that will explicitly be blocked from logging in to Anthill. Leave empty to not block user explicitly. Blocked Patterns will override Allowed Patterns.
9. Test the configuration (required).
- **Test User Name.** Give the user name to test the configuration with. When you click "Save" AnthillPro will verify the configuration. You will be unable to use the integration until this test passes. This can be any user in LDAP.
 - **Test Password.** Give the password for user (given above) to test the configuration with. When you click "Save" AnthillPro will verify the configuration. You will be unable to use the integration until this test passes. This can be for any user in LDAP.
- 10 Click **Save** then **Done** if the test passes.

Using LDAP over SSL with Self-signed Certificate

If you are using LDAP over SSL with a self-signed certificate you will need to import the LDAP certificate to the Java Virtual Machine that AnthillPro runs on. Typically, you can do this in one of two ways:

- **Use a Java KeyStore.** This relies on the Java key manager to supply keys to others as needed, e.g., for use in authenticating the user to others. If you are unfamiliar with Java KeyStores, please see the Java documentation [<http://download.oracle.com/javase/6/docs/>].

Ensure the certificate version can be imported to the JavaVM KeyStore. If the certificate does not import correctly, the integration will fail.

You can verify that the import was successful using other Java-based applications. For example, JXPlorer [<http://jxplorer.org>] can be helpful.

- **Use a Java TrustStore.** Relies on the trust manager to make decisions about who to trust based on information in the TrustStore. If you are unfamiliar with Java TrustStores, please see the Java documentation [<http://download.oracle.com/javase/6/docs/>].

Ensure the certificate version can be imported to the JavaVM TrustStore. If the certificate does not import correctly, the integration will fail.

The certificate must be placed in a directory that AnthillPro can access. Once this is done, the location needs to be added to the `ah3server` start script as follows:

JAVA_OPTS parameter. Use the following parameter with the absolute path to certificate location: `-Djavax.net.ssl.trustStore=/point/to/your/certificate/file`.

You will need to restart AnthillPro for the changes to take effect.

You will also need to use the appropriate LDAP URL syntax (e.g., `ldaps://myldap.server.com:636`) when configuring the LDAP Authentication Realm integration.

RSA SecurID Authentication Realm

The RSA SecurID Realm uses the RSA SecurID token-based system to manage users. You will need access to the System page in order to manage security.

1. Go to **System > Authentication** under the Security menu.
2. On the **Authentication** tab, click the **Create Authentication Realm** button.
3. Check **RSA SecurID** and click **Set**.
4. On the **Authentication** tab, configure Realm:
 - **Name** the Authorization Realm.
 - **Description**. Provide a description of the Authorization Realm.
 - **Authorization Realm**. Select the Authorization Realm this Authentication Realm will use.
This item requires that the appropriate Authorization Realm has already been created. See Configure Authorization Realms.
5. If not setting a **Secondary Authentication Realm**, click **Save** then **Done** to complete. Otherwise proceed to **item 6**.
6. Select the **Secondary Authentication** tab (optional) and click the **Create Secondary Authentication Realm** button.

Upon authentication in the primary Realms, Users are forwarded to this secondary Realm if they are a member of any Roles which require it. The Secondary Authentication Realm adds a second layer of security.

This item requires that the appropriate Roles and Authorization Realms have already been created. See Define Roles and Configure Authorization Realms.
7. Check one of the available Authentication Realms and click **Set**.
8. Configure Secondary Authentication. *Authentication Settings will only take effect if the test login is successful (if test fails, a warning will appear).*
 - **Name** the Secondary Authentication Realm.
 - **Description**. Provide a description of the Secondary Authentication Realm.
 - **Test User Name**. Give the user name to test the configuration with.
 - **Test Passcode**. Provide the passcode to test the configuration with.
9. Click **Save** then **Done**.

Single Sign-On Authentication Realm

The Single Sign-On Authentication Realm relies on an external single sign-on server to handle authentication. Before you can configure authentication, ensure that the appropriate Authorization Realm has been created.

You will need access to the System page in order to manage security.

1. Go to **System > Authentication** under the Security menu.
2. On the **Authentication** tab, click the **Create Authentication Realm** button.
3. Check **Single Sign-On** and click **Set**.
4. On the **Authentication** tab, configure Realm:
 - **Name** the Authorization Realm.
 - **Description**. Provide a description of the Authorization Realm.
 - **User Header Name**. Give the name of the HTTP header that contains the Single Sign-On user name.
 - **Logout URL**. Enter the URL a user is redirected to when they logout of AnthillPro.
 - **Authorization Realm**. Select the Authorization Realm this Authentication Realm will use. If you configured the Single Sign-On Authorization Realm, select from the drop-down. Or, you can use a different tool for Authorization: e.g., the built-in AnthillPro authorization system.
5. Click **Save** then **Done**.

For Single Sign-On Authentication Realm, there is no need to configure a secondary realm.

Add Users

Users are added to the Authentication Realm. Once a user account is activated, AnthillPro users may edit some of the settings configured below (by following the profile link at the top of their browser window). If a user changes his/her contact information or password, AnthillPro will automatically update the changes. See Configure User Profile.

You will need access to the System page in order to manage security.

1. Go to **System > Users** under the Security menu. On the **Users** tab, choose the **Authentication Realm** from the drop-down menu. Click **Set**.
 - If integrating AnthillPro with LDAP, see Configure Authentication Realms. Once configured with LDAP, AnthillPro will utilize users from the LDAP Authentication Realm.
2. Click the **Create User** button. Give the user a **name**, set and confirm a **password**, and check which **roles** the user will have. Click **Save**.
3. Identify the individual user.
 - **Name**. Give the name the user will use for log in. Once this is set, individual users will not be able to change the Name. See Configure User Profile.

- **Roles.** Check the roles this user is assigned to. A user's role determines which Anthill features and projects will be available to them. A new user is automatically given the "user" role. See Defining Roles.
- **First Name.** Provide the user's first name.
- **Last Name.** Give the user's last name.
- **Email Address.** Provide the user's e-mail address.
- **XMPP ID.** If using Jabber, Google Talk, etc., enter the ID where the user will receive notifications from Anthill3. See Configure Instant Messaging.
- **MSN ID.** If using MSN IM, enter the ID where the user will receive notifications from Anthill3. See Configure Instant Messaging.
- **Time Zone.** Select the time zone the **user** is in.
- **Number of Dashboard Rows.** Give the number of recent build to include on the user's dashboard.

4. Click **Save**.

5. To add a new **alias**, click the **User-Repository Alias** button. Select the appropriate **repository** from the dropdown menu and give the user's alias used in that repository. Click the **Add User-Repository Alias** button to finish.

Any source-control user names belonging to this user (i.e., New User) can be mapped in the User-Repository Alias section. If this is done, AnthillPro will then map the changes this user makes. For example, this allows you to only send notifications to those who committed on a certain build, or to those tracking changes and users over time. A mapping is not needed if the source control name is identical to the AnthillPro user name.

Once created, a User-Repository Alias may not be edited. If a user name for a repository has changed, delete the current alias (click the **Remove** icon under the Operations menu) and create a new one.

6. Click **Done**.

Chapter 72. Securing Artifact Sets

Once an artifact set has been created, it is possible to lock down who can access the artifacts within the set. Typically, an artifact set is secured if it contains sensitive information such as production passwords, source code, etc., that only a few people are allowed to access. Once an artifact set has been secured, only the users with the appropriate permissions will be able to resolve/download the artifacts within the set. Users without read permission will generally be able to see the artifact set and the files contained within it, but not perform any actions on the artifacts themselves.

There are two permissions used to secure an artifact set:

- **Read permission.** Allows a user to resolve/download the artifacts associated with a Build Life.
- **Security permission.** For users that *also have security permissions to the project*, the artifact security permission allows users to determine who can set security for the artifact set. For example, users that have been assigned the "admin" role can restrict other roles (i.e., users) from granting security rights.

To secure an artifact set you will need to first ensure that the artifact set default settings are correct, enable artifact security on the server settings and then configure security permissions on the individual artifact set(s).

Securing Artifact Sets Prerequisites

- The artifact set you want to secure must already exist, and you should know which Life-Cycle Model is associated with that Artifact Set. See [Managing Life-Cycle Models](#).
- You must have permissions to modify the Server Settings and Life-Cycle Model configuration on the System page. See [Setting Up Security](#).

Set Default Permissions for Artifact Sets

Before you enable and configure artifact security, you need to verify that the Default Permissions for Artifact Sets are correct. This will ensure that the correct permissions are applied to all existing artifact sets when you enable the system setting.

1. Go to **System > Permissions** from the Security menu.
2. Select **ArtifactSet** from the drop-down menu and click **Set**.
3. On the Artifact Set Permission page, verify that the Default Permissions are correct. The default settings set here will be automatically applied to every existing artifact set -- an action that can't be reversed, even if you disable the server setting.

If you need to change these settings, see [Configure Default Permissions](#) before continuing.

4. Once the Artifact Set Default Permissions are correct, click **Done** and see [Enable Artifact Security](#).

Enable Artifact Security

Once you are satisfied with the Artifact-Set Default Permissions, you can then enable artifact security on the System

page.

1. Go to **System > Server Settings** and select the **Security** tab.
2. Click **Edit** and see **Configure Server Security > Secure Artifact Sets** before continuing.
3. Once enabled, see **Secure Individual Artifact Sets** to manually secure individual artifact sets.

Secure Individual Artifact Sets

Once every artifact set has been assigned the appropriate default permissions and the server setting enabled, you must manually secure select artifact sets at the Life-Cycle-Model level:

1. Go to **System > Life-Cycle Models** from the Project Support menu.
2. Select the Life-Cycle Model which contains the artifact set you want to secure.
3. On the Life-Cycle Model page, select the **Artifact Sets** link.
4. Locate the Artifact Set you wish to secure and select the **View Security** icon (a yellow badge) under the **Operations** menu.
5. Modify the permissions and click **Save**.
6. Repeat the previous steps for every Artifact Set you wish to secure.

Chapter 73. Set Up and Manage Guest Users

The Guest User account gives anonymous access to AnthillPro, and does not require a user name or password at login. Guest users simply open AnthillPro in a browser and the Dashboard page appears. Or, if guest users are sharing workstations with registered users, they follow the 'click here to login as guest' link on the main login page.

Management of guest users is similar to that of other AnthillPro users. However, permissions are not automatically assigned (with LDAP, etc.), and must be manually configured for every instance of AnthillPro. Once the account is activated, guest users are assigned any available role or combination of roles. For example, a User, Build Master, etc., role may be assigned to guest users; or a role created specifically for guest users (named 'guest role' for example) may be assigned. To determine how guest users interact with AnthillPro, each role is then assigned different read and/or write permissions based on a combination of resources and resource types. See Manage Security.

- You must have AnthillPro administrative privileges to configure the Guest User account. See Manage Security.

To activate Guest User Account:

1. Go to **System > Server Settings** under the Server menu.
2. Click the **Security** tab and click **Edit**.
3. Check the **Allow Anonymous Guest Access** box and click **Save**.
4. **Assign Role to Guest User Account.** To use a special 'guest user' role, it must be created prior to completing subsequent items.
 - Go to **System > Users** under the Security menu.
 - Select **System** from the **Authentication Realm** drop-down box and click **Set**.
 - On the **Users** tab, select **guest** from the **Active Users** list.
 - Scroll down and click **Edit**. Assign a **role** for the Guest User account. (The example below assigns a role [named **guest role**] created specifically for guest users.)

Create and modify Anthill users with this form.

Authentication Realm:	System	The Authentication Realm the user will be created in.
Name: *	guest	The username used when logging in.
Roles:	<input type="checkbox"/> Build Master <input checked="" type="checkbox"/> guest role <input type="checkbox"/> System Admin <input type="checkbox"/> User	The roles this user is assigned to. A user's role determines which Anthill features and projects will be available to them. A new user is automatically given the "user" role.

5. Click **Save** when done.

Chapter 74. Perform Security Audits

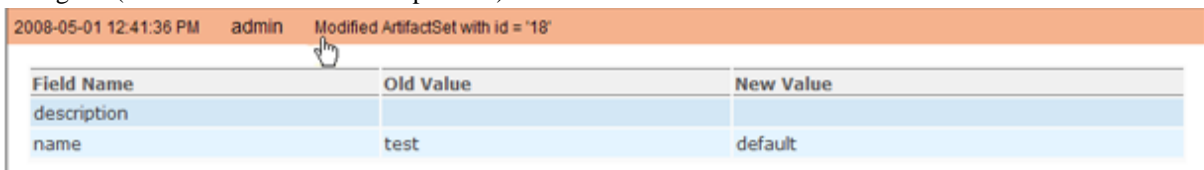
The who-when changes made by Administrative users is recorded and made available on the Audit page (**System > Audit** under the Security menu). When running an Audit, search using a combination of user(s) and date(s).

The audit will return information regarding changes that:

- **Created** a new field, including the date, time, user name, and description (in blue).
- **Modified** an existing field, including the date, time, user name, and description (in salmon).
- **Deleted** an existing field, including the date, time, user name, and description (in pink).

Results are returned from oldest to newest, and are searchable using the navigation menu. Selecting a cell in the table expands the item and provides details regarding the change.

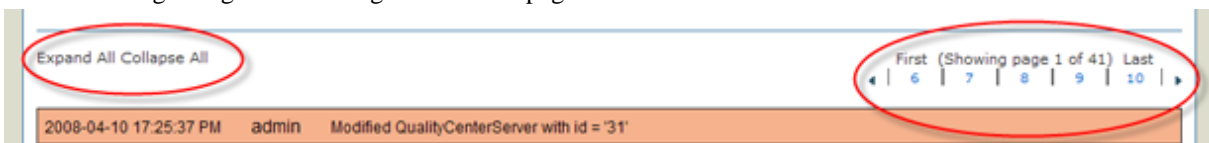
1. Go to **System > Audit** under the Security menu.
2. On the **Audit** page, give the criteria.
 - **User.** Select a user from the drop-down menu. *If no user is selected, the Audit will return results for all users with Administrative permissions.*
 - **Start Time.** Give the beginning start data in the format: 2008-02-31 (e.g., February 31, 2008). *If no start date is set, the Audit will return results for all dates unless an end date is set.*
 - **End Time.** Give the ending start date in the format: 2008-05-09 (e.g., May 9, 2008). *If no end date is set, the Audit will return results for all dates unless a start date is set.*
 - Click **Search**.
3. To view the details of a particular item, click the appropriate cell. To collapse an expanded item, click the colored cell again. (Deleted items cannot be expanded.)



2008-05-01 12:41:36 PM admin Modified ArtifactSet with id = '18'

Field Name	Old Value	New Value
description		
name	test	default

4. Select **Expand All** to view details for every item. To collapse all expanded items, select **Collapse All**. To view other pages: use **Previous Button** and **Next Button**, select a page from the navigation menu, or click **First** to go back to the beginning and **Last** to go to the final page.



5. Click **Done**.