IBM Rational Build Forge Online Help Version 7.0.2.5

Document # SC23-7681-07

Table of Contents

Document # SC23-7681-07

Document # SC23-7681-07

Document # SC23-7681-07

List of Figures

Document # SC23-7681-07

Welcome to Build Forge Help

This help system provides how-to and reference information for the Management Console.

To use the help:

- Select topics in the table of contents at the left of this window. Click on + signs to expand chapters.
- Click the or button in the upper right corner of this window to hide or show the table of contents.
- When you are using the application, click the **Help** button to display context-sensitive help.

Help (HTML) and documentation is available in the **Help** topic:

- **Help** : this help system (HTML)
- **Help** → **Installation** : *Build Forge Installation Guide* as a help system (HTML)
- **Help** → **Installation (PDF)** : *Build Forge Installation Guide* opened in a separate window as a PDF file.
- **Help** → **Help (PDF)** : this help system opened in a separate window as a PDF file.
- **Help** → **What's New (PDF)** : *What's New in the Build Forge System* opened in a separate window as a PDF file.

PDF files are useful for printing multiple topics or for storing for use outside of a running Build Forge system. Help files (HTML) offer more accessibility options. Use your web browser settings to control font size, color, and other accessibility features.

Note

Before using this information and the product it represents, read the information in “Notices for IBM Rational Build Forge” on page 243.

First Edition June 2009.

This edition applies to version 7.0.2 of IBM® Rational® Build Forge® .

Document version and build: 7.0.2-006.0003.

Document # SC23-7681-07

Getting Started

This section provides a quick introduction to the system.

To begin, install the system if you have not already. You can install the system (Management Console and an agent) on a single host.

Standalone Installation

You can try the system with a standalone installation of a Management Console and an agent on the same machine.

If you are installing the system in order to perform a trial or test a specific feature, you may want to install the Management Console and one agent on the same machine. These two components are enough for a complete system, so that you can test the system without using additional machines.

A production installation typically covers many machines, one of which has the Management Console (and optionally, an agent) installed while others merely receive an appropriate version of the agent.

Navigating the Application

This topic provides basic information about using system menus and tabs.

When you use the application:

- Be sure your display resolution is set to at least 1024 x 768. 1280 x 1024 or higher is recommended.
- Do not shrink the browser window smaller than 1024 x 768 (the size it comes up as).
- You can choose an application by clicking on a tab at the top right of the window. For example, click the Console tab to access the Console application.
- The left column of the window has a set of navigation controls:
    - Select a module within an application by clicking the module name in the box at the top left of the window.
    - Three buttons below the module list allow you to show/hide the module list, and expand/hide the left column.
    - Below the module list, the application displays clickable content from the current module.
- Content for the currently-selected module appears in the main content pane. The pane may contain tabs to allow you select among different types of information.

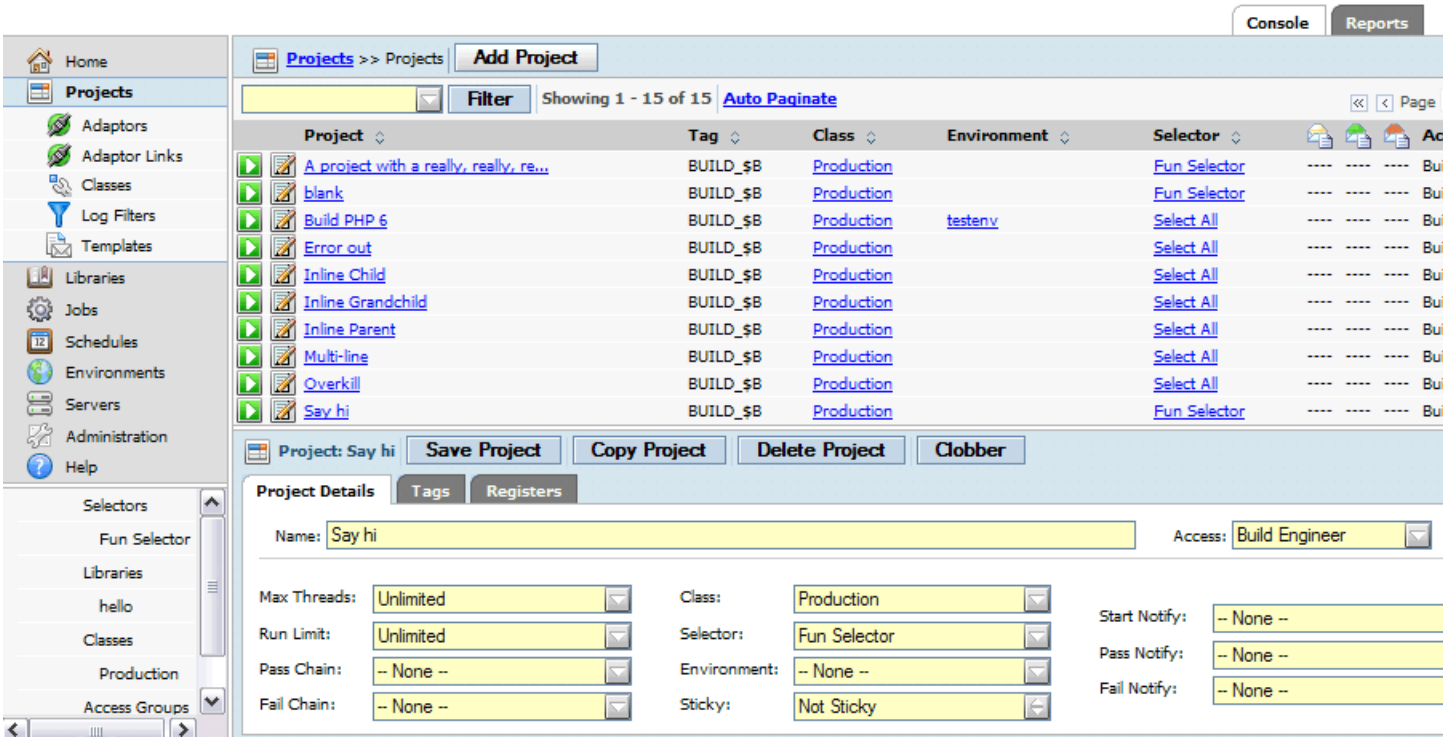
Document # SC23-7681-07

Note

When the main content pane displays a list, the list may be longer than will fit on a single page. Use the page controls in the upper right of the content pane (Page 1 of 4) to navigate the list, or filter the list with the Filter box to show fewer items.

- When you click an object's icon, the system displays the properties for the object. When you click an object's *name*, the system displays the components of the object. For example, in the Projects module, to edit a project's properties, click the project's Edit icon; to view a project's list of steps, click the project's name.
- The system includes buttons labeled **Add** <*object*> and **Save** <*object*>. Clicking a button such as **Add Project** or **Add Step** clears all the information in the object fields so that you can enter new information. Click the **Save Project** or **Save Step** button to save data you have entered as a new project or step.

Major Areas of the Application



- **Module list (top-left navigation pane).** Click a module to move to a different area of the application. The figure shows the Home, Projects, Libraries, and other modules.
- **Module contents (bottom-left navigation pane).** Click an object related to the selected module to view or edit the related object. The figure shows the Selector, Library, and other objects related to the project
- **Application tabs.** Click a tab to change to the Console (the default application) or Report application for the product.

- **Main content pane (upper content pane).** Displays information about the module or object selected in the navigation pane.
- **Object edit form (lower content pane).** Displays details for the module or object selected in the main content pane. The figure shows the details for the project. If the form is empty you can enter values to modify an existing object or create a new one.

Using Filter Boxes

You can quickly filter lists to display just the items you want to work with, by entering a string in a Filter text box.

Filter boxes appear above lists of objects such as projects or servers. When a Filter box is available, you can use it to filter the contents of the list below it. You enter text in the box and the system updates the list to display only items that match your entry.

- Type text in the Filter box and press Enter or click the **Filter** button to list items that match your entry. The system lists only the items that contain the text you entered (in any of their listed columns).

  ### Note

  The filter is case-sensitive.

- You can filter a list based on the contents of a single column by placing the column name in front of the filtering string, like this: "columname:text".
- The system remembers the filter strings you enter. Click the arrow on the **Filter** button to display a list of previous filters entered by you or other members of your organization. You can delete a filter string by clicking the trash can icon to the right of the string.
- The system always provides a **Display All** option in the list of filter strings.

Creating a Hello World Project

This topic describes how to create and execute a simple project to verify that Build Forge is set up properly.

1. Create a server authentication, selector, and a server object.

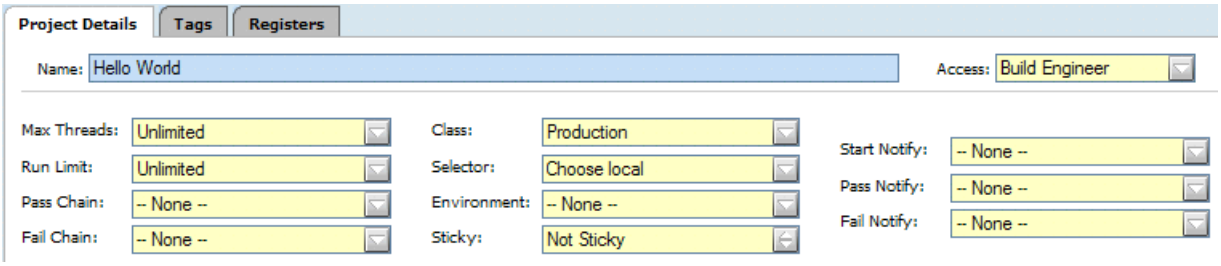
   These objects are required before you can create a project.

2. Create a project named HelloWorld.

   Select **Projects** . In the Project Details form at the bottom of the main content pane, enter HelloWorld as the **Project Name** and choose a selector. Click **Save Project.**

Document # SC23-7681-07

Creating HelloWorld



3. Add a step to the project named EchoHelloWorld.

   a. Select the HelloWorld project. The system displays the empty step list for the project and a blank Step Details form.

   b. In the Step Details form, enter a **Name** of EchoHelloWorld.

   c. In the Command field, enter a command line that will write “Hello World” to standard output on your chosen server.

      For example, the following command line works on Windows®, Solaris, Linux®, UNIX®, and Macintosh OS X systems:

      ```
      echo Hello World
      ```

      Then click the **Save Step** button.

4. Run the project.

   Select **Projects** to redisplay the project list, then click the icon next to the HelloWorld project. The system displays the **Running** tab of the **Jobs** module, with the HelloWorld project listed as running.

   ### Note

   If the HelloWorld project is not listed, skip to step 6.

5. Click the **Refresh** link at intervals until the Hello World project disappears from the Running list.

6. Click the **Completed** tab.

7. Click the job tag for the job.

   The default job tag for an initial job is BUILD_1. The system displays details for the run, with the step log at the bottom of the main pane.

8. Examine the log for the project.

Document # SC23-7681-07

In most Hello World examples, you would see the "Hello World" text in a console window or pop-up window. The Management Console does its work by sending commands to the agent process on the targeted server; the agent then sends the output from those commands back to the Management Console, which stores them in the logs.

The log has many sections; the relevant one is the final EXEC section. Click on it to display the results of your command:

```
EXEC
274 Jun 13, 2006 - 16:52 start [c:\BuildForgeTests\HelloWorld\BUILD_1@mcsystem] echo
 Hello World
275 Jun 13, 2006 - 16:52 Performing variable expansion on command line
276 Jun 13, 2006 - 16:52 Hello World
277 Jun 13, 2006 - 16:52 end [c:\BuildForgeTests\HelloWorld\BUILD_1@mcsystem] echo
Hello World (0)
```

This project demonstrates that you have configured your system correctly, that projects can successfully access a server, run and generate output on a server. The `echo` command can be replaced with any command that can be run on the target server.

Project Samples

Samples of projects are included to help you get familiar with the system.

Project samples are included in the following directory:

```
<bf-install>/samples/projects/
```

To use a sample project, do the following:

- Import it into the Management Console, using the directions in Importing Exported Data Objects.
- Execute the project.

Setting Up Servers

This section describes how to set up and manage Server resources in the Management Console.

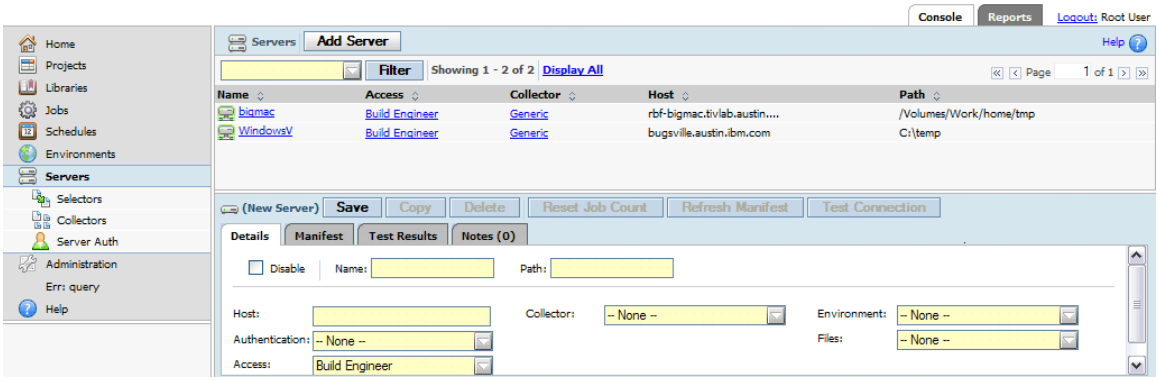
About Servers

In Build Forge, a Server resource is a host set up to run steps.

To set up a machine to be available as server in Build Forge, you must do the following:

- Install an agent on the machine (see the *Build Forge Installation Guide* for more information)
- Create a Server resource using the Management Console

The Servers Module



- Servers have manifests. A *manifest* is a list of server properties. A manifest is populated when a collector runs. If a server does not have a collector assigned to it, a few properties are automatically populated in the server's manifest.
- Manifests are populated by collectors. A *collector* is assigned to a server. A collector both sets property values and collects values for properties from the agent for a Server.
- Projects can use selectors to determine what server will run a step. A *selector* reads server properties from the manifest.

As an administrator setting up the system: first, you create servers. You then create collectors that you can assign to servers. You run the collectors to populate the server manifests. Once that is done, build engineers can create projects that use selectors to determine where project steps run.

Creating Servers

Before you create any projects in the system, you must define at least one server. A server is a machine to which the Management Console can send commands.

Each server must have an agent installed on it. When you add a server, you are describing how the Management Console should access and use a specific machine.

Document # SC23-7681-07

Before you create a server, make sure that the other data objects it depends on already exist. You will need to assign the following items to a server:

- Server authentication: Required. Specifies the login name and password to use with the server.
- Collector: Optional. Defines the properties the system collects from the server, in addition to default properties.
- Environment: Optional. Specifies environment variables to be assigned whenever a project is executed on the server.

Note

If desired, you can create more than one server object within the Management Console for a single physical machine. These are called logical servers; they are typically used to allow projects to access the same hardware with different properties. For example, two logical servers might use different paths or different environments:

- Two logical servers with different paths would create separate working directories on the same machine. You could easily distinguish work performed using one server from work performed with the other because all output would appear in different directories.
- Two logical servers with different environments would execute their steps with different starting environment values.

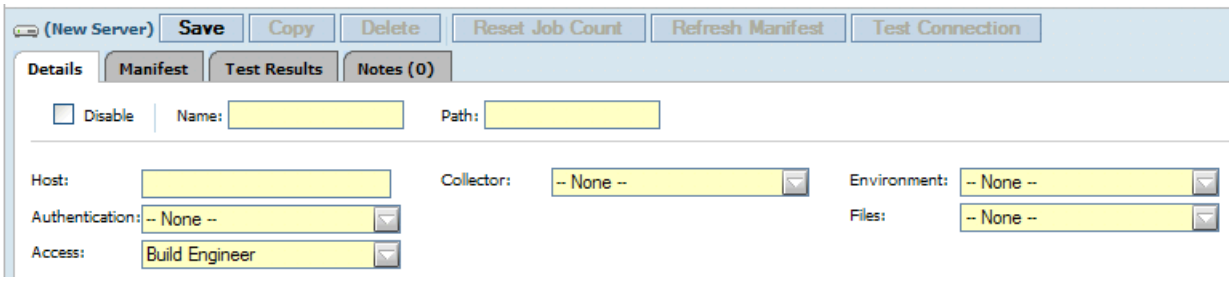
To create a server:

1. Select **Servers.** The system displays the New Server form at the bottom of the main content pane.

   If you have selected an existing server, click **Add Server** to erase the form so that you can add a new server.

2. Fill out the server details:

New Server Form



- In the **Name** field, give the server a name. This name is the BF_NAME property of the server. You reference this name in selectors to choose a specific server by name.

Document # SC23-7681-07

- **Path** : Specifies a directory that the server should use when it creates project and job directories, such as `c:/buildforgeprojects`. The system uses this path value as a starting point when it creates the build directory.

  ### Note

  The system does not create the server path. The path must exist before a build attempts to access the server, or else the build fails.

- **Host** : The host name for a physical machine running a copy of the agent. Use value localhost if you are defining the Management Console machine as a server. (The agent must also be installed on the Management Console.)

  ### Note

  You can include a port number with the host name. For example, <*host_name*>:<*port_number*>. If you specify a port number with the host name, it overrides the port number defined by the Default Agent Port system setting. (See Administration > System > Default Agent Port.)

  ### Note

  Do not precede the host name with a protocol (for example, http://).

- **Authentication** : Select the server authentication to use with this server.
- **Access** : Use this field to select an access group that should be allowed to use this server.
- **Collector** : Select the collector to use with this server.
- **Environment** : Use this field to select a group of environment variables to be applied whenever this server is used to run a project. These variables are applied before all other variables, and should set up parameters specific to the server.
- **Files** : Use this property to define the types of file transfers allowed on this server via the .get and .put commands. You can choose to allow no transfers (None), file reads (.get), file writes (.put), or both (.get and .put). This setting must be set appropriately for .get and .put commands to work.

3. Click **Save** . Your new server appears in the server list at the top of the content pane.

   To verify that you have correctly configured the server, select your server in the list and then click **Test Server** . The system reports errors if it cannot communicate with the server.

Server Authentication

Use server authentications to associate login credentials to a server. You can use the same credentials for many servers, and update the credentials globally, by managing a set of server authentications.

Document # SC23-7681-07

A server authentication stores a login name and password as a single named object which you can associate with one or several servers. Use the Server Authentication page to create and edit server authentications.

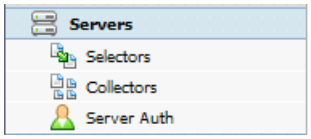
Creating Server Authentications

Use server authentications to store the login information for sets of servers.

Each server needs a server authentication assigned to it so that the Management Console can log in to the server with appropriate privileges. Server authentications separate the login information from the server records so that you can apply the same login information to more than one server.

1. Select **Servers** → **Server Auth** .

Selecting Server Authorizations



The system displays the list of existing server authentications at the top of the main content pane, and a blank Server Auth Details form at the bottom.

Note

If you select a server authentication, the system populates the Server Auth Details form with the selected server's authentication information. To empty the form so that you can create a new authentication, click the **Add Server Authentication** button.

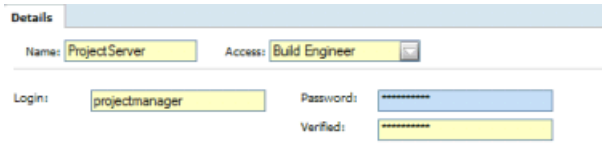
2. At **Name** , enter an authentication name, a logical name to identify the server authentication in the system.

3. At **Login** , enter the server login name.

Note

If the user name is from a domain user, you must include the domain in this field. For example, MYDOMAIN/joeuser.

Server Auth Details



4. At **Password** , enter the password.

5. At **Verified** , reenter the password.

6. Click **Save Server Authentication** .

The system stores a new server authentication with the name you selected.

Overriding Server Authentication

You can force the system to use your Management Console login credentials instead of the server authentication assigned to the server, via a special environment variable. To override the normal authentication, add a variable named _USE_BFCREDS, with a value of 1, to an environment used by your project or step. If you add the variable to the project environment, the system uses the override on every step in the project.

When the system attempts to run a step whose environment contains _USE_BFCREDS=1, the system uses the console login credentials of the user who started the project to execute the step's command.

Note

If you are using LDAP/Active Directory Authentication, the **Store User Authentication Locally** system setting must be set to Yes (its default value) for the _USE_BFCREDS function to work. When the setting is Yes, the system caches user authentication information in encrypted form, and can then access the user authentication information for use with _USE_BFCREDS. Otherwise, the system does not store the LDAP information and cannot use it.

Allowing Use of Restricted Server Auth

Use the Execute Inaccessible Server Auths permission to allow a user to execute a step on a server with a server auth to which they do not have access.

As a prerequisite, the user must already have or be granted access to the server (Servers > Access).

The server auth is simply a login used to access a server. Server auths are associated with an access group (Servers > Server Auth > Access).

You might create the following server auths for a server:

- a dev/dev server auth and associate it with the Developer access group for the server
- a qa/qa server auth and associate it with the QA access group for the server (the QA access group is user-created and in this example has the same default permissions as the Developer access group)
- a prod/prod server auth and associate it with the Build access group for the server

In the example, to allow a user who has access only to the qa/qa server auth to be able to run a step as the prod/prod server auth, add the Execute Inaccessible Server Auths permission to the QA access group.

Document # SC23-7681-07

Note

If the user has access to the server but does not have access to the server auth through the Execute Inaccessible Server Auths permission, the step will still run but only if the environment variable _USE_BFCREDS is set. For details, see Overriding Server Authentication.

Selectors

Server selectors allow you to describe the kind of server that a project or step should use by listing desired properties and values. When you apply a selector to a project or step, the system uses the selector to determine which servers are valid choices for the task, and then selects an available server from the valid ones.

You can use selectors to be specific (choosing a specific server by name) or general (any Windows® server). A selector is a list of properties which describe the desired server. To manage selectors, use the **Servers** → **Selectors** page.

If you want to select servers based on properties you define, create appropriate collector properties *first*. For example, you can create a collector with a property named BUILDING (with a value equal to the name of the building that houses a server). This allows you to select servers based on their physical location. However, when you create selector variables, you must choose properties from a list that the system generates from all of your collectors.

If a selector *does not* find a server that matches its property list, then the project or step fails and the system creates a build note.

A selector contains a list of property/value pairs called *variables*. For each variable, you can specify a value and a comparison. For example, you could specify a property "CompilerVersion = 1.1" to select only servers that have that property, but you could also specify "CompilerVersion >= 1.1" to select servers with versions 1.1, 1.3, 2, and 2.0. Selectors support numeric and string comparison operations.

- A variable can be required or optional. When multiple servers match the required variables, the system picks the one that matches the most optional variables.
- You can repeat optional variables in a selector, to increase the score of a server that matches them. For example, you might require MEM_TOTAL> = 1GB but repeat MEM_TOTAL >= 2 GB three times to bias the system to choose servers with memory of at least 2 GB. See below for the details of how the system makes its choice.
- You can use the .include statement to add environment variables from another selector. The .include statement references an environment. If there are duplicate variables in environments, the system counts each instance of the variables when it assigns scores to servers.

To choose a server, the system:

1. Compiles a list of the servers which contain all the *required* variables in the selector.
2. Rates each server, granting the server a point for each *optional* variable it matches.

Document # SC23-7681-07

    - If the selector contains more than one copy of the same variable, the extra copies grant extra points to servers that match them.
    - The system assigns one extra point to the server with the lowest BF_LOADRATIO value.

3. Chooses server that received the most points.

Although it does not appear in a server's manifest, the property BF_NAME is automatically assigned to every server; its value is the logical name of the server. To create a selector for a specific server, create a selector variable that selects BF_NAME=<*logical server name*>.

When comparing selector variables with manifest properties to look for matches, the system looks at the variable's value and the manifest property's value. The system performs a lexical (string) comparison unless **both** values match the following criteria for numbers:

- If the value starts with a digit and contains only digits and decimal points followed by at least one digit, the system performs a numeric comparison.
    - 5, 5.5, 0.5, 5.0, and 5.5.5 are considered numbers.
    - 5., .5, 5.., 5..5, 5.4.6_05, and 5.6i5 are all considered strings
- A numeric value containing more than one decimal point causes a sub-version numeric comparison, where the system compares each decimal-separated section. While 5.21 is less than 5.3 (ordinary numeric comparison), 5.21.0 is greater than 5.3 (sub-version numeric comparison).

Note

The system always performs a string comparison if you use the “contains”operator. Also, the “contains” operator is case-insensitive.

Selector/Manifest Comparison Examples

Property Name	Manifest property value	Operator	Selector Variable Value	Comparison Type	Match?
PerlVersion	v5.8.4	>=	5.2.1	Lexical	Yes
PerlVersion	v5.8.4	>=	v.5.2.1	Lexical	Yes
PerlVersion	v5.8.4	>=	v5.22.1	Lexical	Yes
OS_VERSION	1.15	>=	1.1	Numeric	Yes
OS_VERSION	1.10	>=	1.1.0	Sub-version numeric	Yes
BF_NAME	WinServer1	contains	win	Lexical	Yes
BF_NAME	Server123	=	123	Lexical	No

Creating Server Authentications

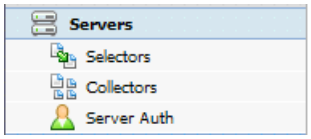
Use server authentications to store the login information for sets of servers.

Document # SC23-7681-07

Each server needs a server authentication assigned to it so that the Management Console can log in to the server with appropriate privileges. Server authentications separate the login information from the server records so that you can apply the same login information to more than one server.

1. Select **Servers** → **Server Auth** .

Selecting Server Authorizations



The system displays the list of existing server authentications at the top of the main content pane, and a blank Server Auth Details form at the bottom.

Note

If you select a server authentication, the system populates the Server Auth Details form with the selected server's authentication information. To empty the form so that you can create a new authentication, click the **Add Server Authentication** button.

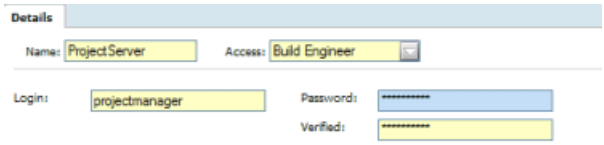
2. At **Name** , enter an authentication name, a logical name to identify the server authentication in the system.

3. At **Login** , enter the server login name.

Note

If the user name is from a domain user, you must include the domain in this field. For example, MYDOMAIN/joeuser.

Server Auth Details



4. At **Password** , enter the password.

5. At **Verified** , reenter the password.

6. Click **Save Server Authentication** .

The system stores a new server authentication with the name you selected.

Document # SC23-7681-07

Collectors

You define collectors to determine what properties the system collects from (or assigns to) your servers. The collector assigned to a server is like a blueprint for the server's manifest.

The Collectors section of the Servers module (**Servers** → **Collectors**) lists the available collectors and allows you to create new collectors.

A collector consists of a series of properties which are assigned to any server that uses the collector. However, the specific values of the properties can vary from server to server, because a collector is a set of instructions for collecting data.

You can define several types of properties in a collector:

Set Value
: These properties simply assign a named, static value to the server. Specify the property name and the value. The same value is assigned to all servers that use the collector.
The system recognizes values with special names as defining behavior for a server. These special values begin with the underscore (_) character. See “Special Set Value Properties” on page 27 for a list of these.

Built-in
: These properties return information about the server. For a list of built-ins, see “Built-in Properties Reference” on page 226. When you choose this type, the system allows you to pick a built-in from a list.

Run Command
: For these properties, you define a command for the system to run. The command's output is used to populate the property value in the manifest. You can use a regular expression to specify what part of the command's output to use.
To define a Run Command property, fill out the following fields:

  Property
  : The name of the property, for the manifest.

  Command
  : A command to run on the server.

  Regular Expression
  : A Perl regular expression. Optional. If specified, the system tries to match the regular expression against *each line* of output from the command. The first time a line matches, it retrieves the value of $1 (a Perl convention), and uses it as the value for that property. The regular expression must include at least one set of parentheses so that it returns a value. Consult Perl documentation for more information on constructing Perl regular expressions.

    ### Note

    If you do not specify a regular expression, the system uses *up to the first 255 characters* of output from the command to populate the property in the manifest.

Document # SC23-7681-07

.include — These allow you to nest collectors. When you create a .include property, you specify the name of another collector as its value. When it creates or updates the manifest, the system inserts the properties from the referenced collector.

Note

The system applies collector properties in the order they are listed in the collector, and later properties of the same name *override* earlier ones. Use this feature when you include one collector within another one. If you want to use some of the properties of a collector but not all, override the ones you do not want to use.

The system also applies a few properties automatically, such as the BF_NAME property that contains the logical name of the server. See “Special Manifest Properties” on page 26.

Testing Servers

You can perform a set of diagnostic tests on a server. Select **Servers** → **<server_name>** and then click the **Test Server** button.

If the server fails the test, try one of the following:

- Verify the username and password in the server authentication you are using for the server
- Verify that you are using the correct host name
- Reinstall the agent on the server, or verify that it is installed

Enabling and Disabling Servers

You can temporarily disable a server in the system. When a server is disabled, the system does not run jobs on it.

To disable or enable a server:

1. Select **Servers** .
2. Click on the server whose status you want to modify. The system displays the details for that server.
3. Click the **Disable** check box in the Details form at the bottom of the main pane.
4. Click the **Save Server** button.

Updating the Server Manifest Immediately

On the Servers page, use the Refresh Server Manifest button to update the server manifest for the selected server immediately.

This selection overrides the following system settings that control how frequently server manifest properties get updated:

- Active server refresh interval
- Inactive server refresh interval

1. Select **Servers** .
2. Select a server.
3. Click **Refresh Server Manifest** .

Updating the Job Count for a Server

On the Servers page, use Reset Job Count button to reset the job count (BF_JOBS) for the selected server to zero. BF_JOBS is the number of steps or jobs currently running on the server.

This selection allows you to reset BF_JOBS if it does not correctly reset when a job completes, fails, or is cancelled.

For example, cancelling multiple jobs occasionally fails to reset BF_JOBS. If BF_JOBS is not reset it can reach the limit for the default_MAXJOBS system settings causing steps or jobs to not run.

1. Select **Servers** .
2. Select a server.
3. Click **Reset Job Count** .

Updating the Job Count for All Servers

In Administration > System Settings, use Reset Server Job-Count to simultaneously reset the job count (BF_JOBS) for all servers to zero. BF_JOBS is the number of steps or jobs currently running on the server.

The reset occurs when the manifest check interval runs. (The default time is every 10 seconds.)

After BF_JOBS has been reset for all servers, the Reset Server Job-Count value reverts back to No (the default).

1. Select **Administration → System** to display the list of system configuration settings.
2. Locate Reset Server Job-Count.
3. Click **Reset Server Job-Count** .
4. On the Details tab, select **Yes** as the value.
5. Click **Save**.

Selecting the Server for a Project or Step

Use a selector to choose the server for a project or step.

Document # SC23-7681-07

A selector describes the kind of server that is appropriate for the project or step. It can specify a server directly (by name) or indirectly (by an attribute collected by a collector and stored in the manifest).

The following are common best practices for setting up selectors:

- By Name: create selectors named after each of your servers which specify the server by its BF_NAME (the unique name used in the Management Console). Choose these selectors when you want to be specific about your choice.
- By Server Pool: if you organize servers into named pools, create a collector for each pool that assigns the pool name as a property. Then create selectors for each pool name.
- By Server Attributes: you can choose servers based on their real functional properties, such as available hard-disk space, operating system, or number of CPUs. To implement dynamic selection, first create collectors that collect and assign appropriate properties. Assign the collectors to the appropriate servers. Finally, create a selector for each property or set of properties that might influence your choice of machine.
  - A selector for machines whose operating system name includes “Windows® ”
  - A selector for machines with more than one CPU
  - A selector for machines running at less than a specified load

  Each selector chooses from all servers that match its criteria.
- Nested collectors: you can use .include properties to nest collectors within one another. You may want to create individual collectors for each server, for example, so that each server can have some unique properties that you specify, while using .include to include some utility collectors. For example, you might create a collector called Version that issues commands to check the version numbers for some key tools in your environment, such as Perl and Java, and then include that collector in all of your server-specific collectors.

Dynamic Server Selection

Instead of choosing servers directly, you can set up data to describe the right *kind* of server for a project or step.

You create collectors to attach properties to servers, and those properties are stored as the server's manifest. You define selectors to specify desired properties for a project or step. Server manifests allow the system to choose the right server for a project or step dynamically, based on the current situation.

Simple Example: You create a selector named Mercury which selects servers whose BF_NAME is equal to Mercury. When you run the project, the system selects the server named Mercury. Because this property is unique, the system always chooses the same server, and if that server is not available, the project fails for lack of an available server.

More Complex Example: You create a collector named Stats that collects RAM, number of CPUs, and hard-disk space available. You assign the collector to several servers, one named Mercury

Document # SC23-7681-07

(with 512MB RAM), one named Mars (with 1GB RAM), and one named Jupiter (3GB RAM). Then you create two selectors named MuchRam (which selects servers with at least 2GB RAM) and NotMuchRam (which selects servers with at least 256MB RAM). Finally you create two projects, each of which uses one of these selectors.

When you run the projects, the system chooses the server Jupiter for the project that uses the MuchRam selector, because it is the only one that matches. The project that uses the NotMuchRam selector may end up with any of the available servers.

Later you add a server, Neptune (2GB RAM), to the system. The next time you run a project that uses the MuchRam selector, the system may choose either Neptune or Jupiter. If Jupiter is down for some reason, the system uses Neptune, as it is the only one left that fits the selector.

Three different data objects allow the system to dynamically choose servers:

- A *collector* is an object that defines the set of properties that the system collects from or assigns to a server. The system refers to a collector when it checks a server's properties; then it stores those properties in the database as a manifest.
- A *manifest* is a list of the properties for a specific server. Where a collector specifies a property like memory size or Perl version as something to collect or assign, a manifest stores the result of the collection operation.
- A *selector* is a list of properties and comparisons like MEM_TOTAL = 512. The system can compare the properties of a selector with a manifest to see if a server meets the requirements for a particular selector.

Manifests

A *manifest* for a server is the set of values that the Management Console collects from or assigns to that server and stores as a record in the database.

When it chooses a server for a project or a step, the system compares a selector against its set of manifests and picks a matching server.

You can view the manifests for your servers in the Servers module. Select **Servers** → **<server_name>** , then click the **Manifest** tab.

You cannot directly change the manifest for a server. Instead, you must edit the collector assigned to the server, or assign a different collector to the server. The collector defines the kinds of properties the system assigns to a server or attempts to collect from it; the manifest is the resulting set of property values.

Set How Often a Server's Manifest is Updated

You can use system settings to control how often the system queries a server for its manifest properties.

By default, the system checks Built-in and Set Value properties more frequently than Run Command properties. It uses several system settings to determine when to perform updates:

Document # SC23-7681-07

- The settings **Active server refresh interval** and **Inactive server refresh interval** set the number of seconds between updates of Built-in and Set Value properties. The settings discriminate between active (currently running a project) and inactive (not currently running a project) servers, so that you can have properties refresh more frequently on servers when they are being used.
- The setting **Default _AGE** sets the number of seconds between updates of Run Command properties. These properties, which require that the system run a command on the server to get updated data, create more network traffic during their updates. The default value for this setting is 86400, which provides for updates once per day. IBM does not recommend setting a value lower than this setting unless the values you retrieve through Run Command properties change frequently.

You can use a collector to set a value for _AGE that applies only to the servers that use that collector. When you use the special name _AGE for a variable, the system interprets its value as the desired number of seconds between updates. To set this up:

1. Select **Servers** → **Collectors** .
2. Select an existing collector, or create a new one.
3. Add a property to the collector.
    a. Select the type Set Value.
    b. Enter the **Property** name _AGE.
    c. Enter a **Value** in seconds.
    d. Click the **Save Variable** button.
4. If necessary, select one or more servers and change their Collector properties to match the collector you just edited.

Limiting Concurrent Jobs on a Server

Use a collector to specify the maximum number of jobs the system allows a particular server to run simultaneously, and set the default maximum via a system setting.

The system limits how many processes it tries to run on any one server. If the server has a _MAXJOBS property, the system limits the number of processes to the value of _MAXJOBS. If the server does not have a _MAXJOBS property, the system uses the default value, which comes from the **Default _MAXJOBS** system setting. To change the default value, select **Administration** → **System** → **Default _MAXJOBS** and set a different value.

Note

Other programs can run on the server. The system only limits the number of processes that the Management Console runs on the server.

Document # SC23-7681-07

To give a server a non-default value, create a collector that includes a _MAXJOBS variable, and then assign the collector to the server. When you use the special name _MAXJOBS for a variable, the system interprets its value as the maximum number of processes for the server. To set this up:

1. Select **Servers** → **Collectors** .
2. Select an existing collector or create a new one.
3. Add a property to the collector.
    a. Set the **Type** to Set Value.
    b. Enter the **Property** name `_MAXJOBS`.
    c. Enter a **Value** equal to the number of processes you want to allow on the servers that use this collector.
    d. Click the **Save Variable** button.
4. If necessary, select one or more servers and change their **Collector** properties to match the collector you just edited, so that they use the new collector.

Built-in Properties Reference

The Management Console collects built-in properties from servers and then assigns the values to the server manifest.

List of Built-in Properties

Built-in properties are used by several different data objects in the system:

- **Selectors** can use built-in properties as selector variables, to match servers with certain values in those properties.
- **Collectors** use built-in properties to collect data from servers.
- **Manifests** store the values of built-in properties when they have been collected.

Built-in properties are not automatically added. You must add a built-in property to a collector for the property to display in the manifest.

Document # SC23-7681-07

Built-in Properties for Collectors and Manifests

Property	Description
CPU_ARCH	The returned value is a `label` for an architecture name, as shown: • `HP-PA`: HP Precision Architecture • `IA-64`: Intel Itanium • `MVS`: IBM S/390 • `PPC`: PowerPC • `PPC-64`: PowerPC 64 • `SPARC`: Sun SPARC • `X86`: x86-compliant architecture used by Intel, AMD, Cyrix, and others.
CPU_LOAD (Windows only)	For Windows systems, the CPU load (or CPU usage) is expressed as a percentage of capacity (between 0 and 100).
CPU_LOAD1 (UNIX/Linux only)	For UNIX/Linux, the average number of processes (load average) running or waiting to run over the last minute. CPU_LOAD1 is a measure of CPU activity. An idle computer has a load number of 0. Each process that is using CPU or waiting for CPU adds to the load number by 1.
CPU_LOAD5 (UNIX/Linux only)	For UNIX/Linux, the average number of processes (load average) waiting to run over the last 5 minutes as reported by the operating system. CPU_LOAD5 is a measure of CPU activity. An idle computer has a load number of 0. Each process that is using CPU or waiting for CPU adds to the load number by 1.
CPU_LOAD15 (UNIX/Linux only)	For UNIX/Linux, the average number of processes (load average) waiting to run over the last 15 minutes as reported by the operating system. CPU_LOAD15 is a measure of CPU activity. An idle computer has a load number of 0. Each process that is using CPU or waiting for CPU adds to the load number by 1.
CPU_MHZ	Processor speed in Megahertz. Certain conditions have to be met for this property to be filled in successfully: • Linux: frequency scaling must be enabled. • Windows: the ~MHz registry entry must exist and be filled in. • x86 and x86-64 processors: inline assembly must work.

Document # SC23-7681-07

Property	Description
CPU_MANUFACTURER	Company name of the processor manufacturer. The names are assumed based on architecture if the information is not directly available. No value is returned if there is insufficient processor information available. Supported values are: • `AMD`: for their x86 and AMD64 processors • `Cyrix`: for their x86-compliant processors • `DEC`: for Alpha and VAX • `HP`: Hewlett-Packard Precision Architecture • `IBM`: IBM S/390 and PowerPC G5 • `Intel`: Intel x86 (including Intel64), IA-64 Itanium • `Motorola`: PowerPC G4 • `NexGen`: x86-compliant processors • `National`: National Semiconductor x86-compliant processors • `Rise`: Rise x86-compliant processor • `Sis`: Sis x86-compliant processor • `Sun`: Sun Microsystems SPARC • `TransMeta`: TransMeta x86-compliant processor • `UMC`: UMC x86-compliant processor • `VIA`: VIA Technologies x86-compliant processor

Document # SC23-7681-07

Property	Description

CPU_MODEL	Manufacturer-specific CPU model numbers. The following values are reported:
	• x86 architecture
	  • 386
	  • 486
	  • 586
	  • 686
	  • X86_64
	• PPC architecture
	  • 6xx
	  • POWER
	  • RS64
	  • G3
	  • G4
	  • G5
	  • Cell

CPU_SERIAL	Serial number of the CPU or machine. Currently this functionality is limited to the following architectures: • x86: Intel or Transmeta serial numbers only. Note: most x86 processors will not report a serial number. No value is returned in those cases. • MacOS/X: the serial number assigned is retrieved from an I/O registry. Requires that CoreFoundation and IOKit frameworks be found.
DISK_FREE	For UNIX/Linux, the amount of free space (in MBs) on the file system specified by the server Path property. For Windows, the free disk space (in MBs) on the drive specified by the server Path property. For example, 4096 MB for systems with 4 GB of free disk space.

Document # SC23-7681-07

Property	Description
DISK_TOTAL	Total free disk space available. This value is reported for the agent's base path, which may have a separate allocation that is smaller than the entire remaining disk or partition. Disk space management varies significantly between operating systems.
MEM_LOAD (UNIX/Linux only)	For UNIX/Linux, the amount of RAM or system memory currently in use is expressed as a percentage of total real memory (between 0 and 100).
MEM_FREE	The amount of RAM or system memory (in MBs). For example, 1183 MB for systems with 2 GB of RAM.
MEM_PAGESIZE	The RAM or system memory page size (in MBs). The standard page size for the host system. For example, 4096 MB is a 4 KB page size.
MEM_TOTAL	The total RAM or system memory (in MBs). For example, 2048 MB for a system with 2 GB of RAM.
NET_FQDN	Fully Qualified Domain Name (FQDN) of the machine where the agent is running. It is reported based on the address that the agent is using to communicate. The address returned can be an IPv4 or IPv6, based on the address actually being used. See also NET_IPV, NET_IPV4, and NET_IPV6.
NET_HWADDR	Hardware address for the interface reported in NET_IFACE.
NET_IFACE	Name of the interface used by the agent. • Windows: the name as reported by the `ipconfig` command, for example `Intel(R) PRO/100 VE Network Connection - Packet Scheduler Miniport` • Other systems: the name as reported by `ifconfig`, for example `en0` or `eth0` or `OSA1`.
NET_IPV	The type of IP connection used to communicate with the agent, either 4 for IPv4 or 6 for IPv6.
NET_IPV4	The IPv4 address used by the agent to communicate. On connections over IPv6, if the agent is able to identify an IPv4 address for the same interface, that address is reported.
NET_IPV6	The IPv6 address used by the agent to communicate.
NET_SPEED (Windows only)	Windows: speed of the interface in Mb/sec, for example `1000` for Gigabit Ethernet.
NUM_CPU	The number of CPUs on the machine.
OS_HOSTID	Result of the `gethostid()` system call. Normally this is not very informative unless a system administrator has set `/etc/hostid` to an informative value.
OS_SYSNAME	The operating system name of the server. For example, Windows XP, AIX, or Macintosh OS, and so on.

Document # SC23-7681-07

Property	Description
OS_RELEASE	The operating system release level of the server. For example, if the server OS is Microsoft XP Version 5.1.2600, this number is 5.
OS_VERSION	The operating system version of the server. For example, if the server OS is Microsoft XP Version 5.1.2600, this number is 1.
WIN_SERVICEPACK (Windows only)	The number of the Windows service pack installed on the server. For example, 2 for Service Pack 2.

Special Manifest Properties

Some manifest properties are provided automatically by the system. Unlike built-in properties, special properties do not have to be added to a collector in order to become part of a manifest.

You can use special manifest properties in selector variables to choose servers based on some dynamic properties, such as how many jobs the server currently has.

Document # SC23-7681-07

Special Manifest Properties

Property	Description
BF_AGENT_VERSION	Version number of the agent installed on the server.
BF_JOBS	Number of jobs (steps) running at the same time on the server. This value is updated *every time the console assigns a step to the server*, independent of other manifest property updates.
BF_LAST_REFRESH	Time of the last update of built-in properties in the manifest, as a UNIX®-style time stamp (number of seconds since January 1, 1970).
BF_LASTJOBS	Number of jobs running on the machine the last time the manifest was refreshed.
BF_LAST_UPDATE	Time of the last update of run-command properties to the manifest, as a UNIX®-style time stamp (number of seconds since January 1, 1970).
BF_LOADRATIO	Specifies a calculated value: the number of jobs (BF_JOBS) divided by the maximum number of jobs allowed for the server (_MAXJOBS setting). A server that has 1 job running and _MAXJOBS=4 has a load ratio of .25.
BF_NAME	Used as a selector condition. It specifies the server to run on. The value is the logical name of the server in the Build Forge system. The BF_NAME property is not displayed in the manifest list.
BF_RESERVE	Used as a selector condition only. It is a flag that takes no operator or value. If present in a selector, a slot is reserved on the selected server for the duration of the job. • If a step in job specifies a different server to run on, then the slot on the selected server continues to be reserved while that step executes. • If a step specifies the selected server explicitly, the reserved slot is used for that step. This flag protects projects from a delay caused by losing their slot on a server when one or more of their steps execute on other servers.

Special Set Value Properties

Some Set Value properties, when named in a collector, cause behavior in the system. These properties begin with the underscore ("_") character. The system uses the values of these properties to apply behavior to servers which acquire these properties from a collector.

Note

You cannot create properties which begin with the string "BF_" as these names are reserved for use by the system.

Document # SC23-7681-07

Special Set Value Properties for Collectors and Manifests

Property	Description
_AGE	Defines how often a manifest should be refreshed, in seconds. The default value, 86400, provides a refresh once per day. A value of 3600 causes the system to update the manifest every hour. (You can modify the default setting via the system setting **Default _AGE**).
_MAXJOBS	Defines the maximum number of jobs (steps) the system should run on the server at one time. Default value is 3; you can modify the default using the system setting **Default _MAXJOBS** .

Copying Files to and from Servers

You can use the .put and .get commands to copy files from one server to another during a project. This section describes how to enable file copying and how to use the .get and .put commands to move files around.

Note

These two commands return an error if the destination file already exists; you cannot use them to replace files. Also, the commands can only move single files. To move whole directory trees, see the .rget and .rput commands (“.rget” on page 103 or “.run and .runwait” on page 103).

Enabling File Copying on a Server

The default settings for servers do not allow files to be copied via dot commands. Before you attempt to use the .get or .put commands with any server, enable the commands on that server by changing the **Files** property for the server.

To change the Files property for a server:

1. Select **Servers** → **<ServerName>** to display the properties for the server.
2. Select a value other than None for the Files: property. You can enable copying files from a server (.get), to a server (.put) or both.

Getting a File from a Server

Use the .get command to get a copy of a file from a server and place it in a destination relative to the current step's working directory. For example, if you have a server named winbuildserver1 with a file config.txt in its config directory, you can add the following step to your project to copy the file to the current server's config directory:

```
.get winbuildserver1:./config/config.txt ./config/config.txt
```

For more information on using .get, see “.get” on page 94. Also see the description of paths in “Working Directories for Jobs” on page 129 .

Document # SC23-7681-07

Putting a File on a Server

When you want to copy a file from your current server to a different server, use the .put command. The following example step assumes that you have a config.txt file in a config directory on the current server, accessible from the current path:

```
.put ./config/config.txt winbuildserver1:./config/config.txt
```

For more information on using .put, see ".put" on page 102. Also see the description of paths in "Working Directories for Jobs" on page 129.

Using Servers at Other Sites

With the system in place, you can immediately access any servers on your local network for which you can get the necessary logins. But in an organization with servers at more than one site, you can extend your reach even farther:

- Use a Virtual Private Network (VPN) connection to access a server at another site. If a remote office has a Solaris server and you need to perform a Solaris build, you can access one without buying your own hardware.
- Provide access to the Management Console to developers or testers at remote sites, and they can launch builds on machines in your office, or view reports to track down problems.

Subscribing to RSS Data Feed for Server Status

The Management Console runs a server status check to verify that the server can pass a functional test and verify that the agent can log in. The Test Results tab displays the results of the status check. The Management Console automatically checks servers status whenever a server is created or edited and you can initiate a server status check at any time, for example, before running a project.

The Build Forge RSS data feed for server status displays the same information as the Test Results tab in the Build Forge Management Console.

To subscribe to the RSS data feed for server status, do the following:

1. In the Build Forge Management console, select **Servers** .

   The Web browser detects the RSS feed and displays an RSS icon in the browser address bar.

2. In the RSS aggregator tool, load the Build Forge RSS data feed.

   For example, copy the URL to add it to the list of RSS data feeds or drag-and-drop the RSS icon to add the URL to the list of RSS data feeds.

3. Subscribe to the RSS data feed to save the URL and be notified when updates occur.

Document # SC23-7681-07

Note

- For details about loading URLs and subscribing to RSS data feeds, consult the documentation for your RSS aggregator tool.
- To view Build Forge system messages or server status through an RSS data feed in languages other than English, your RSS aggregator tool must support UTF-8 multibyte character encoding.

Document # SC23-7681-07

Working with Environments

This section describes how to set up and manage Server resources in the Management Console.

About Environments

Environments are applied to servers, projects, and steps.

The Environments Module

The Build Forge® system allows you to manage environment variables separate from the projects and servers that they apply to. This paradigm provides you with a great deal of flexibility in creating the environment for any particular command:

- You can create environments which contain environment variables.
- You can use the .include command to nest environments together (see ".include" on page 95).
- You can assign one environment to each server, one to each project, and one to each step within a project.

When the system runs a step, it applies all the relevant environments to create the step environment: server, project, and step, in that order.

Note

Too many environment variables in a single project can cause Build Forge to hang. To avoid this problem, limit the number of environment variables to fewer than 8000. You can also use .source to set large numbers of environment variables without impacting performance. See "Running Scripts Before a Command With .source" on page 113.

Variable Syntax

You can use UNIX®-style or Windows®-style variable syntax in step commands or environment variables definitions.

Important

Variable names may contain only alphanumeric characters (a-z, A-Z, 0-9) and the underscore character (_).

The system translates UNIX-style $VAR or Windows-style %VAR% syntax into an appropriate format for the server assigned to run the command or process the environment variable.

Document # SC23-7681-07

During translation each side of the assignment is run against the target environment to create the final expression to be executed.

1. The preparser evaluates the variable assignment. Special characters are consumed unless escaped by the backslash character ($, %, {, }, ", '). If preparsing is turned off, all characters are passed.

2. Each side of the variable assignment is evaluated by the target environment.

3. The evaluated variable assignment is executed.

The parser, the Windows environment, and the various UNIX and Linux shells interpret special characters differently. Take care when using special characters and the backslash escape character.

Note

The system cannot change the syntax if the variable is hidden. See "Variable Actions" on page 33.

Variable Values

Set a value for each variable added to an environment. The set value is the default. You can override the default value by assigning a new default prior to running the project or by temporarily changing the default as a project start option.

Review the following information about setting environment variables:

- **Special characters**: do not use the %, $, [,], {, }, ", or ' characters in values.

- **List of values**: Define a list of possible value for a variable by separating values with the pipe (|) character. Build Forge saves the values as selectable options in a list. The first value in the list becomes the default value for the environment variable.

  Note: If you reference substitution variables in the list, add a space before the substitution symbol ($). The space causes the variable value to display instead of the substitution variable. ($BF_TAG in the following example.)

  ```
  TEST= $BF_TAG|FOO or TEST=FOO| $BF_TAG
  ```

- **Dot commands as values**: some dot commands can be used as the value of an environment variable; in these cases, the system replaces the dot command with other values. See "Environment Dot Commands" on page 113.

- **Carriage returns**: variables do not store carriage returns. You can assign a multiple line value to a variable, as shown in the following example. The contents of the file `text.txt` is assigned to the variable *test*:

  ```
  .bset env "test = `type text.txt`"
  ```

  Build Forge concatenates the lines in the file. For example, suppose the file's contents were as follows:

Document # SC23-7681-07

```
A first line
And a second line
```

The variable's value then becomes the following:

```
A first lineAnd a second line
```

Variable Actions

When the system applies a variable to set up the environment for a server, project, or step, it can do more than simply assign a value to it. You can choose an **Action** for each environment variable. The following actions are available:

- Set: The default option. Assign As causes the system to assign the value you specify to the variable, creating the variable if it does not already exist.
- Set if not set: This action assigns the value to the variable only if the variable does not already exist on the server when the system tries to set it. See “How the System Applies Environments to Steps” on page 34 for more information.
- Append: The system appends the value specified to any existing value for the variable.
- Prepend: The system prepends the value to the current value.
- Clear: The system clears the variable, setting it to null. The contents of the Value field are not used.
- Delete / Unset: The system deletes the variable on the server. The contents of the Value field are not used.
- Assign Hidden: The system assigns the variable, but hides the value in the logs, showing it as “*****”. Use this option when you are including a password or other sensitive information in a variable; for example, if you need to include password information in an _MAP variable but do not want to make the password visible to users who run the project.

Note

The system normally changes the syntax of a variable in a command line to the appropriate form for your operating system (%VAR% for Windows®, $VAR for Linux®/UNIX® systems). It cannot do this for a hidden variable.

Special Behavior on Projects

When you create an environment variable, you can define some special behavior that takes effect when the variable is assigned to a project. Use this behavior to make your environment variable groups more flexible, creating groups with some variables which are ignored, read-only, required, or hidden when the group is assigned to a project.

To use the special behavior, set the **On Project** property when you create or edit an environment variable.

Document # SC23-7681-07

The On Project property offers the following options:

- Normal: The variable behaves normally when assigned to a project.
- Required: The system prompts for a value for the variable if no default value is defined.
- Read-Only: The value of the variable cannot be changed when the group is assigned to a project.
- Suppress Display: The variable is not displayed on the project Start page, although the value still exists and can be used in steps.
- Must Change: You must change the variable's value when you launch the project; if you do not, the system prompts you to change it.

How the System Applies Environments to Steps

This section describes how the system applies environments.

Before the system executes a step, it applies all the relevant environments for the step, to create the step environment. It evaluates server, project, and step environments, in that order. This has the following ramifications:

- The step gets the environment variables for the server it actually runs on.
- Project variables can override server variables, and step variables can override project variables. The last value set wins. You can use variable actions to change this behavior.

For example, if the variable X is defined with the standard **Assign As** action, but with different values, for a server, project, and step, the step environment wins, because it is the last one applied. However, if the project uses the **Assign If Not Assigned** action, then its value is ignored, and if the step uses the Append Existing action, then the final result becomes the combination of the server and step values.

- If the step's commands change variable values, those values persist only during the current step. The system resets the values from the database when it creates the environment for the next step, so if you want to persist changes, you need to use a dot command (.set or .bset) to change the value in the database.
- Since a user can modify project variables when launching a job, the system maintains a job environment, a copy of the project environment that reflects any user changes made when starting the job. However, if the step's environment is set to the same named environment as the project (not just default), the system refers back to the master record for the environment, and gets the original variable values.

For example, if you launch a project with a project environment named Java that includes a variable JavaVersion = 1.4, and you edit the value to 1.5, any steps that use the default environment get the value 1.5, while any steps that reference the Java environment specifically get the original value of 1.4.

Document # SC23-7681-07

Note that when the system starts a job, it copies the project environment variables to a database record set aside for the job, and refers to this job environment thereafter when getting project default values. If the user modifies the starting values of any project variables when the user starts the job, those values are what is recorded in the job record.

Environment Variable Inheritance in Chained Projects

When a project is launched through a chain, the system applies environment variables from the calling project. They are applied differently depending on whether the called project is called as an Inline chain or as a conditional chain (Pass Chain or Fail Chain).

- Inline chain: the steps of the chained project are executed as though they are *part of* the calling project. If Step 1 in Project 1 calls Project 2, variables are applied in this order:

  1. Server environment for Project 2, if specified.
  2. Project 1 environment
  3. Step 1 environment
  4. Step environments for each step in Project 2, if specified

  In each step, a variable's value is controlled by the last environment to set it.

- Pass Chain or Fail Chain: the chained project works as if *nested within* the calling project.

  - User-defined variables are passed to the called project using the same names.
  - System-defined variables (BF_ prefix) are passed from the calling project created in the called project's environment using the prefix BF_CALLER. For example, the calling project's BF_PROJECTNAME is passed into the called project's environment under the name BF_CALLER_PROJECTNAME.
  - System-defined variables (BF_ prefix) are created for the called project.

  The system applies all the environments that are relevant. The called project sets up variables from the calling project's environment and its own environment in the following order:

  1. Called project server environment
  2. Calling project's variables, in a set, with BF_ variable names changed to BF_CALLER.
  3. Called project server environment (applied a second time in case it was modified by the caller's variables)
  4. Called project environment
  5. Step environments (if specified) as they are executed

Document # SC23-7681-07

Using the .set and .bset Commands to Change Variables

Since the .set command modifies the master database record for an environment variable, a .set command that modifies a project variable has no effect on the current job, but can be used by succeeding jobs when they make their own copies of the project variables. If a .set command changes a step environment variable, succeeding steps see those changes because those steps retrieve their values from the master database record. The .set command cannot act on variables that do not already exist when the command executes.

The .bset command modifies the job record; use it when you want to modify the project environment variable values for the current job only. The .bset command has no effect on the master database record for a variable. You can also use the .bset command to create a new variable during the job.

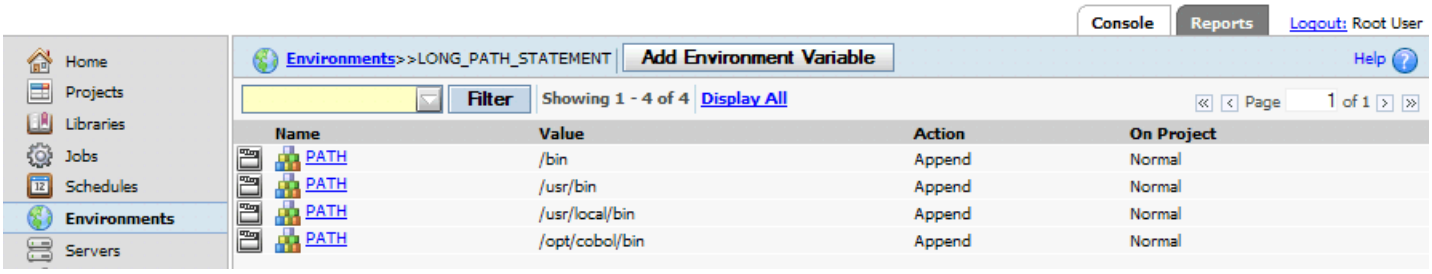
See “Changing Environment Variable Values During a Job” on page 111 for more information on using the .set and .bset commands.

Storage Limits for Environment Variables

The value specified for an environment variable cannot be longer than 255 characters. If an environment variable has more than 255 characters, create multiple assignment statements for the variable and select Append in the Action field to create a single statement. The environment variable whose values you want to append together must belong to the same environment.

The following figure shows multiple assignment statements for the PATH environment variable.

Long Environment Variable Joined with the Append Action



The following code example shows the combined value for the PATH environment variable.

```
PATH = /bin:/usr/bin:/usr/local/bin:/opt/cobol/bin
```

Creating Multiple Values for a Variable

Define multiple possible values for an environment variable by separating values with the pipe (|) character. The first value in the list is default value. The values you provide are displayed as selectable options in a list box for the environment variable.

To create a value list for an environment variable, do the following:

Document # SC23-7681-07

1. Select **Projects → Environments.**
2. In the list, select the environment.
3. Click **Add Environment Variable** .
4. At Name, enter the environment variable name.
5. At Value, enter two or more values separated by the pipe (|) character, as shown in the following example:

   ```
   value1|value2|value3| $value4
   ```

   - The first value in the list (value1) becomes the default.
   - If one of the values is represented as a substitution variable ($value4), include a space before the substitution ($) character.

     Adding a space displays it as $value4 in the list of values. If you want to display the variable value, do not include a space.
6. At Action, select **Set** .
7. At On Project, select **Normal** .

Note

If the list of values exceeds the 255-character limit, you can create multiple environment variables to hold the values. See Creating a Long Value List for a Variable.

Creating a Long Value List for a Variable

If the list of values for an environment variable exceeds the 255-character limit, you can:

1. Create multiple environment variables (each less than 255 characters) to hold the values.
2. Create a separate variable to join the multiple-variable lists.

Note

Do not use the ${varname} variable syntax with this feature, and make sure the `.include` occurs before the combined variable.

To create long value list, do the following:

1. Create an environment to manage the environment variables. In this example, the environment is called *MyLongValueList*.
2. Create as many environment variables as you need to hold the values. Each variable must be 255 characters or less.

Document # SC23-7681-07

Separate individual values in the list using the pipe (|) character, as shown for environment variables A and B:

- A = “value1|value2|value3”
- B = “value4|value5|value6”

Note

Optionally set the **On Project** property for A and B to **Suppress Display** to prevent them from being displayed and selected when the project is run.

3. Use the .include command to add the environment *MyLongValueList* to your project's environment:

```
.include = "MyLongValueList"
```

Note

Make sure the `.include` is listed before the combined environment variable in the project environment.

4. Define a new environment variable in your project environment to combines the value lists in environment variables A and B.

```
LongList = $A|$B
```

5. Execute the project. When you execute the project, the system combines the values from and A and B and lists them in the Value list box.

Use Uppercase Variables with Cygwin

IBM recommends that you name environment variables in all uppercase when you intend to use them with a system running Cygwin.

When you use the Cygwin interface to create a UNIX®-like shell environment on Windows®, be aware that Cygwin converts all of your environment variables to uppercase names. This is a normal convention for Cygwin, but it may cause confusion if you define mixed case or lowercase environment variable names within the system, then find that when accessed from Cygwin, the names are all uppercase.

System-Defined Variables (BF_ Variables)

The system sets values for the following variables automatically in every step. The variables are listed here in alphabetical order.

Note

Environment variables starting with “BF_” are reserved for use by the system and cannot be defined by the user.

Document # SC23-7681-07

Variable	Contains
BF_AGENT_PLATFORM	A string identifying the platform that the agent is running on.
BF_AGENT_VERSION	The version number of the agent for the current server.
B	The default tag variable, which starts at 1 and gets incremented for every job. Can be used in tags.
BF_BID	Contains the job ID number, unique for jobs of the current project.
BF_CLASS	The class for the project
BF_D	Date. Can be used in tags. Format is determined by the Tag: Date Format system setting.
BF_HOST	The TCP/IP host name of the logical server.
BF_J	Day of the year. Can be used in tags.
BF_LASTGOODRUN	The date of the last passing job of the same project, or the last job if no passing job exists.
BF_LASTGOODTAG	Tag for the last passing job (or last job, if no passing jobs stored of the same project).
BF_LASTGOODUNIX	Same as BF_LASTGOODRUN, but expresses the date in UNIX® format.
BF_LASTRUN	The date of the previous job of the current project.
BF_LASTTAG	The tag string for the previous job of the same project.
BF_LASTUNIX	Same as BF_LASTRUN, but expresses the date in UNIX format.
BF_ONFAIL	The Halt/Continue flag for the step
BF_PID	Contains the project ID number.
BF_PROJECTNAME	The project name.
BF_PROJECTNAME_PHYS	Project name as used to create the project directory. The system changes characters specified in the **Invalid Relative Dir Characters** system setting into underscore characters to create the project directory. For example, if the setting includes a space, then a project named “My Project” receives a project directory named “My_Project”. that
BF_ROOT	The base working directory for the job.
BF_SERVER	The logical server name
BF_SERVER_ROOT	The base path assigned to the logical server
BF_SID	The step ID, which uniquely identifies the current step in the project.
BF_SPID	Contains the calling project ID if the current project was called by another project; otherwise, the value is the same as BF_PID.
BF_STEPNAME	The step name.
BF_SSID	The step ID of the calling step, if the current project was called by another project; otherwise, the value is the same as BF_SID.
BF_T	Time. Can be used in tags. Format is determined by the Tag: Time Format system setting.

Document # SC23-7681-07

Variable	Contains
BF_TAG	The expanded tag string generated when the job was started.
BF_TAG_PHYS	The tag string with spaces replaced by underscores, except in steps with the Absolute option selected, for which it is the same as BF_TAG.
BF_USER	The user name of the owner of the job
BF_W	Day of the week (a numeric value, from 0 to 6).

Special Case Variables

The system watches for some special environment variable names. When a step's environment contains one of these variables (either specifically or inherited from a project or server), the system performs actions. The following table describes these variables.

Variable	Contents
_CI_BUILD_DELETE	Set this variable to any value to delete the build and associated build data after the job runs. (The tag variable is reset to its initial value, prior to the deleted build, if no other project builds executed.)
_CI_BUILD_KEEP	Set this variable to any value to keep the build and associated build data after the job runs. For example, if your job includes an adaptor link and the adaptor step fails, the other project steps do not execute. You might want to keep a copy of the build records for the job, for example, for debugging.
_CLEARCASE_VIEWS	Specifies a list of ClearCase views to start before command execution. Set the value to a comma-separated list of views; for example, “View1,View2,View3”.
_CLEARCASE_VOBS	Specifies a list of ClearCase VOBs to mount before command execution. Set the value to a comma-separated list of VOBs; for example, “\Vob1,\Vob2,\Vob3”.
_CONTEXT_LOG_RANGE	Use this variable to limit log output to lines near filter matches. It takes a positive integer value, and causes the system to omit log output except for a range of lines around each filter string hit whose size is equal to the variable's value. For example, if you set the variable to 5, your logs will show only lines with filter matches, plus the 5 lines preceding and 5 lines following those matches.

Document # SC23-7681-07

Variable	Contents
_ERROR_THRESHOLD	Establishes the maximum numbers of errors (caught by the “Set Fail” filters you have defined) allowed. Using this variable, you can establish thresholds for individual steps and/or for a project, and either fail the step or project when the threshold value is met, or merely note the fact that the threshold was met in the job notes. When you set the _ERROR_THRESHOLD variable for a project, you can use one of the following forms: • A value 5 or F5 indicates a maximum of 5 failures are allowed before failing the job. • A value like N7 indicates that the system should add a message to the job notes that the threshold was met when 7 errors occur. When you use the variable in a step, the system counts the errors in the individual step. Additional forms are available: • A value like W9 indicates that after 9 errors, the step is put in a warning state, regardless of future errors caught by filters. • A value like C8 indicates that after 8 errors, the step is set to failure status, but any Clear Fail filter can clear the failure. **NOTE:** The errors counted by this variable are defined as strings that match filters with Set Fail actions and which are assigned to steps in the project. Each string identified as a failure by a filter counts as one error toward the step total, and one toward the project total.
_EXITCODE_MAP	Specifies a list of numbers (separated by commas, spaces, semicolons, or colons) that the system should accept as indicators of step success. By default, an exit code of 0 indicates success; when this variable is specified, any values listed in it also indicate success.
_InterfaceLoggingLevel	Controls how much log data Build Forge logs when it runs an adaptor step. Create an environment variable (in your adaptor environment) with the name _InterfaceLoggingLevel. Assign it an integer

Document # SC23-7681-07

Variable	Contents
	value from 0 to 8. Logging levels are inclusive, for example, level 2 includes information from levels 1 and 0. • 0: Exec line plus server connection errors or cancel notification; nothing else • 1: Parsed commands (commands as they will be sent to the server) • 2: Unparsed commands (commands prior to having their local variables set) • 3: Build and environment variable SET lines • 4: Temp and internal variable SET lines • 5: Environment evaluations, e-mail group additions, BOM text logging lines • 6: Block & Sub-block start/end lines • 7: (Default logging level) Agent output that will be checked against match patterns, plus the lines that matched the patterns. • 8: All agent output
_LOG	Specifies a path name to create a copy of the command output in. Use this variable to save a copy of the job log on the server. If the file exists, the system appends to it.
_MAP	See “Windows Mapped Drives” on page 45 for a discussion of how to use this variable.
_NO_PREPARSE_COMMAND	The system normally attempts to resolve the values of environment variables before sending commands to agents. When the _NO_PREPARSE_COMMAND variable is defined (with any value), the system sends variables to agents without resolving them. Use this variable to ensure that your operating system shell handles the variables.
_PRISM_DIR_POSTCMD	Used with plug-ins for IDEs. Specifies a command to be run on directories after the project step has executed. See “Special Variables for Test Projects” on page 219 for more information.

Document # SC23-7681-07

Variable	Contents
_PRISM_DIR_PRECMD	Used with plug-ins for IDEs. Specifies a command to be run on directories before they are copied to the server for a project step. See “Special Variables for Test Projects” on page 219 for more information.
_PRISM_FILE_POSTCMD	Used with plug-ins for IDEs. Specifies a command to be run on files after the project step has executed. See “Special Variables for Test Projects” on page 219 for more information.
_PRISM_FILE_PRECMD	Used with plug-ins for IDEs. Specifies a command to be run on files before they are copied to the server for a project step. See “Special Variables for Test Projects” on page 219 for more information.
_SUPPRESS_LOG_OUTPUT	When defined (with any value), causes the system to omit nearly all of the log output normally produced by the agent. Some Management Console log messages remain, and filter matches are also shown.
_TIMEOUT	A value that overrides the Timeout property for one or all of the steps in your project.
_TRAP	A string to be executed if the current step fails; the string can be set to the name of an executable file or command. **NOTE:** The output of the command is not returned to the console because the connection between the console and the agent is closed when the step fails; if you want to retain output from a command executed via _TRAP, have the command write its output to a file for later retrieval.
_USE_BFCREDS	When set to 1, the system uses the *user's* login credentials to log in to servers, rather than the credentials stored in the server authorization attached to server. The system uses the Management Console login credentials of the user who started the project to execute the commands in the project. You can set this variable for a single step, or for an entire project. **Note** If you are using LDAP/Active Directory authentication, the **Store User Authentication Locally** system setting must be set to Yes (its default value) for

Document # SC23-7681-07

Variable	Contents
	the _USE_BFCREDS function to work. When the setting is Yes, the system caches user authentication information in encrypted form, and can then access the user authentication information for use with _USE_BFCREDS.

Setting the Adaptor Log Level

Control how much information is written to the step log for the adaptor by using the _InterfaceLoggingLevel environment variable.

1. Add _InterfaceLoggingLevel to the adaptor's environment.
    - Level 8 logs the most information and level 0 logs the least.
    - Logging levels are inclusive. For example, level 2 includes information from levels 1 and 0.
    - Level 7 is the default logging level.
2. Assign a log level as the value for the _InterfaceLoggingLevel variable:

0: Exec line plus server connection errors or cancel notification; nothing else

1: Parsed commands (commands as they will be sent to the server)

2: Unparsed commands (commands prior to having their local variables set)

3: Build and environment variable SET lines

4: Temp and internal variable SET lines

5: Environment evaluations, e-mail group additions, BOM text logging lines

6: Block and sub-block start/end lines

7: (Default logging level) Agent output that will be checked against match patterns, plus the lines that matched the patterns

8: All agent output

Resolving Undeclared Environment Variables

To reference environment variables in step commands and environments, Build Forge supports the substitution variable formats (shown with the echo command) in the following table.

If the referenced variable is undefined, how it is resolved depends on whether pre-parsing is turned on (default) or off. Pre-parsing behavior is set by editing the no_preparse command in the

Document # SC23-7681-07

bfagent.conf file or the _NO_PREPARSE_COMMAND environment variable. See Special Case Variables.

If the variable is undefined and pre-parsing is on, Build Forge replaces the variable with its name. The exception to this rule is $[foo] which replaces an undefined variable with an empty string.

Substitution Variable Format	Undefined Variable Resolution - Preparsing On (Default)	Undefined Variable Resolution - Preparsing Off
echo %foo%	foo	Windows: %foo% UNIX or Linux: blank
echo $foo	foo	Windows: $foo UNIX or Linux: blank
echo ${foo}	foo	Windows: ${foo} UNIX or Linux: blank
echo $[foo]	blank	Windows: $[foo] system error

Windows Mapped Drives

Windows® 2000 and XP operating systems manage mapped drives differently. Windows 2000 uses a common namespace for drive mappings while Windows XP uses separate namespaces for each user session. The agent attempts to remap remembered connections for user accounts, but may not be able to successfully complete the mapping at runtime. You can use a special environment variable to assist with drive mapping on Windows: the _MAP variable. When you set this variable, the Windows Agent maps drives before executing your steps.

A typical practice when using the _MAP variable is to assign it in the project environment, so that the same drive mapping is passed down to all the step environments through environment variable inheritance. Note that if you also define a _MAP variable in a step environment, the step environment's value overrides the project environment, as only one _MAP value can be defined for a particular step.

Although it is intended for Windows environments, use forward slashes to separate directory path names in the _MAP variable. When the paths are used, the agent automatically corrects them as needed.

For example, setting _MAP to

```
X:=//server/share
```

defines a runtime mapping that connects the X: drive to the Windows UNC path name \\server\\share.

Multiple drives can be mapped by providing additional mapping specifications in the _MAP variable, with semicolons to separate them:

```
X:=//server/share;Y:=//server/share2
```


Document # SC23-7681-07

By default, drive mappings on Windows are performed using the same login/password as defined for the logical server. You can map a drive for a different user by adding the username and password in parentheses after the mapping, as in the following example:

```
X:=//server/share(alternateusername,password)
```

Note

If you must use special characters for passwords, (%, $, [,], {, }, ", or ') escape them with a \ character. Example: `\$`

Drives mapped via the _MAP variable are unmapped on command completion.

Even if they map successfully, drive mappings on Windows 2000 may still be inaccessible if a user logged onto the system's console is using the drive or share in question.

Selecting the Next Available Drive Letter

You can have the system pick the next available drive letter. Use the following syntax:

<driveletter>?=//<directory path>

For example, you can set _MAP as follows:

```
X?=//server/share
```

In this case the system does *not* map a drive to X. Instead, it maps to the next available drive, and stores the *drive letter it selected* in a variable named _MAP_X; if the selected drive is F, the _MAP_Y variable's value is “F:”. You can use the variable to access the mapping.

You can use any letter you like, and you can even use multiple mappings, as follows:

```
X?=//server/share; Y?=//server/public
```

If this example is run on a machine whose next available drive letter is F, it creates the following mappings:

- F: mapped to //server/share
- G: mapped to //server/public

The example also creates the following variables:

- _MAP_X with a value “F:”
- _MAP_Y with a value “G:”

Note

If you use the next available drive syntax when targeting a Windows® system that uses Cygwin, you must escape the question mark with a backslash as follows:

```
Y\?=//server/share
```


Document # SC23-7681-07

Agent-based Drive Mapping

You can also map drives via a configuration parameter in the agent. The *map* parameter, when added to the `BFAgent.conf` file, uses a syntax identical to that of the *_MAP variable*. You can use this parameter to create drive mappings for specific servers. If you also use the _MAP variable, its mappings are applied after the agent parameter's mappings.

Document # SC23-7681-07

Working with Projects

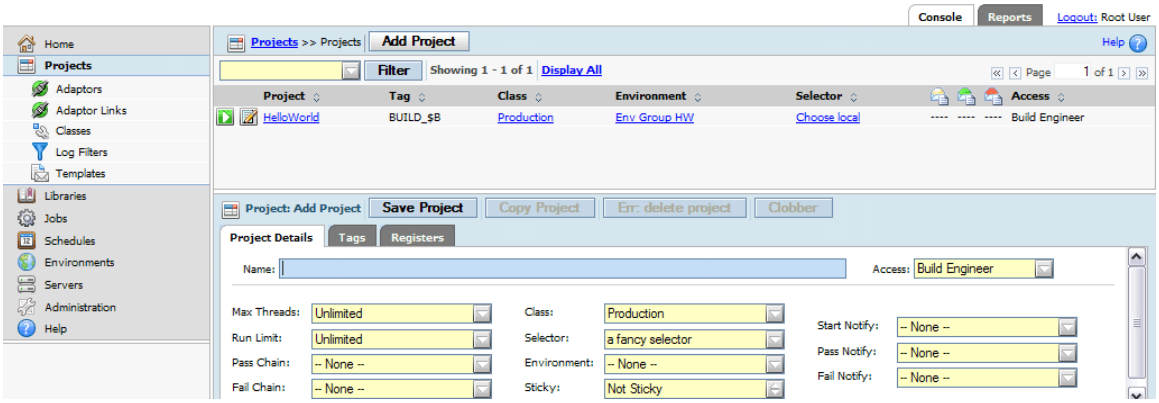
This section describes how to create and manage projects in the Management Console.

About Projects

Use the Projects module to create new projects and edit or view existing projects. Projects are executable sets of steps, with their own environment group and server properties.

Within the Projects module you can view a project, edit a project, or add a new project.

The Projects Module



Changing Project Properties

To change project-level properties, select **Projects** , then click the icon next to the desired project's name.

Project Name

The name of the project; used to refer to the project in lists and in the database.

The project name is used to construct the project directory when the project is executed. The system allows you to use any characters you like in the project name; it's up to you to make sure the project name does not contain characters that are invalid. Since a project may contain steps that run on different operating systems, avoid special characters and symbols in your project names.

Access

The access group allowed to view and use the project. The **Access** property is used along with permissions to determine what a user can do. For example, to launch a job, you must be a member of the access group specified for the project, and you must also be a member of a group that has the Execute Jobs permission. See “Security Overview” on page 151 for more information.

Document # SC23-7681-07

Tag Format	A string that defines the tags for the project, using plain text and tag variable references. See “Tagging Jobs Dynamically” on page 122 for more information on tag formats.
Tag Sync	Synchronize the tag variables for two projects. Select the project whose tag variable you want to synchronize with the current project. When two projects are synchronized, their variables are drawn from the same pool, so that when they run in sequence, one project gets the value 1, the next gets the value 2, and so on. See “Synchronizing Tags” on page 125.
Max Threads	The maximum number of parallel processes the project is allowed to launch. Use this field to keep a project from using too many system resources. Each thread-enabled step and any inline projects (which themselves might launch thread-enabled steps) can result in parallel processes, but all of those processes are counted against the maximum for the parent project. The system stops launching new parallel processes when it reaches the Max Threads value, and waits until the number of parallel processes for the project drops below the Max Threads value before continuing. See “Threading: Running Steps in Parallel” on page 78 for more information on threading.
Run Limit	The **Run Limit** property sets the maximum number of jobs of the project that are allowed at one time. If you launch a project, but the currently active jobs already equal the limit, the new job stays in the Waiting queue until one or more of the jobs completes. If a schedule attempts to launch a project when the number of running projects equals the run limit, the system does not launch a new job at all. Also, projects that are launched via an inline chain are not considered instances of the original project and do not count towards its run limit.
Class	Each project must be assigned to a class, which assigns global properties to groups of projects. See “About Classes” on page 62 for more information.
Selector	The name of the selector to use when choosing a server for the project. The system uses this selector as the default for any steps within the project that do not specify their own selectors. See “Selectors” on page 12. If a selector is not specified, the project is added to the Libraries module instead of the Projects module. Library projects use the selector of the calling project.
Pass/Fail Chain	Select the project that is executed when the project build passes or fails. Setting a pass/fail chain at the project level allows you to invoke separate pass/fail actions based on the pass/fail status of the project. This capability is similar to setting pass/fail

Document # SC23-7681-07

	actions at the step level within a project. At the project-level, the pass/fail actions are triggered by the project run status not the step status.
Environment	An environment to apply after the Server environment and before the Step environment. For more information on how environments work together, see “How the System Applies Environments to Steps” on page 34.
Sticky	Enable the Sticky check box to force all the steps of the project that use the default project server to stay on the same server, and to wait for it to become available if it is busy. For more information on this option, see “Making Steps Stick with a Server” on page 51.
Start Notify, Pass Notify, Fail Notify	Use these fields to direct the system to send notification e-mails on project start, pass, and/or fail, by selecting an access group in one or all of these fields.

Running Projects

There are several different ways to launch a project.

This task assumes that you have already created a selector, server, and project.

- While viewing the list of projects, click the Quick Start icon in front of any project to start the project immediately.

  You cannot use this method if a project has no steps, or if it has any environment variables with the **Must Change** On Project action. Running a project this way uses its default values for selector, class, tags, and environment variables.

- While viewing the project's steps, click **Start Project** .

  This method displays the Start Project page for the project, where you can change project parameters, environment variable values, and select steps to exclude from the run:

  - Select new values for project parameters.
  - Edit the project tag variable values.
  - Edit the project environment variable values. If you want your changes saved as the new defaults for these variables, click the **Save Environment** check box.
  - Select the Steps tab to display the list of project steps. You can select individual steps to exclude them from this run only.

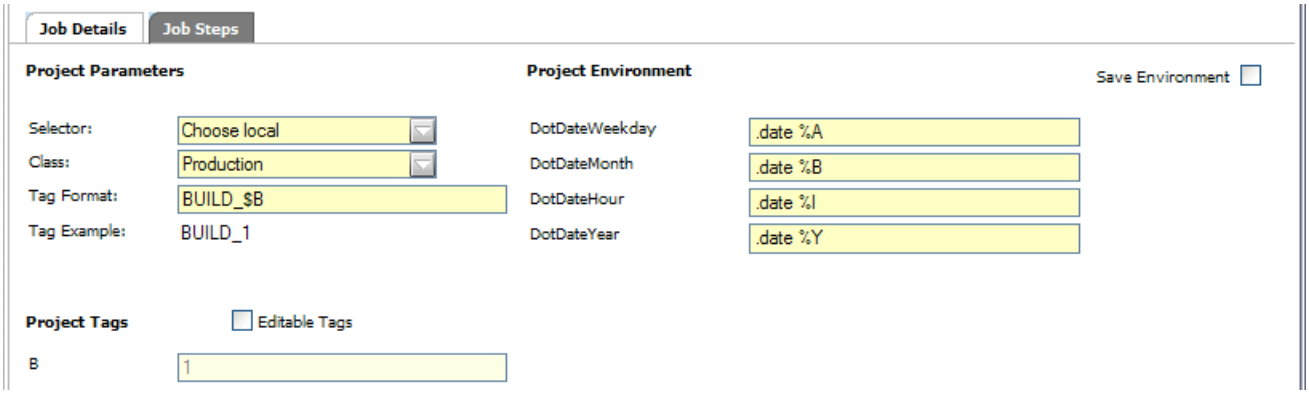
  When you have made your choices, click the **Execute** button to start the project.

- Select **Jobs** → **Start** and then click the project name.

  As when you use the **Start Project** button, this method displays the Start Project page.

Document # SC23-7681-07

Start Project Page



While a project is running, view the **Jobs → Running** page to check the project status.

To view job results, select **Jobs → Completed** to display the completed jobs. Click the Tag Name to access options for viewing job results.

Copying a Project

To make a copy of an existing project, display the existing project's list of steps by selecting **Projects → <ProjectName>** , then click the **Copy Project** button.

Select similar options from the **Library** module to make a copy of a library.

The system displays a dialog box to prompt you for a name for the new project or library.

When you copy a project or library, the system copies the following aspects of the existing object to the new object:

- The steps and all of their properties
- All the project properties listed on the **Project Details** tab, such as the project's tag format, class, selector, and other properties.

The system does not copy the following properties:

- Tag variables (found in the **Tags** tab on the project properties form)
- Project registers (found in the **Registers** tab on the project properties form)

Making Steps Stick with a Server

Steps within a project can run on different servers if their selectors allow it. But you may want all or most of the steps of a project to run on the same server, whether or not you specify that server in advance. The project-level **Sticky** property gives you that option.

Document # SC23-7681-07

To view project-level properties, select **Projects** , then click the icon next to the desired project's name.

The Sticky property applies only to the steps in a project which do not specify a selector of their own. If a step has a selector option other than None, the system uses that selector to choose a server for the step—even if the selector is the same as the project's selector.

When the Sticky property is set, the project uses the same server for every step whose selector field is set to None.

When the system starts an inline project, it chooses a server for the project based on the inline project's settings. The Sticky property of the calling project does not affect the inline project, and the inline project obeys its own Sticky property if it is set.

When the system starts an inline library, it obeys the following rules:

- An inline library, with Sticky property not checked: uses the selector of the calling step as the default selector for the inline steps.
- An inline library, with Sticky property checked: uses the *server* of the calling step as the default server for the inline steps.

Note

You can use the .bset server command to change the default server for a project during the job. Steps that occur after the .bset command use the new default set by that command, and stick to that new server.

Setting Pass/Fail Chains for a Project

Setting a pass/fail chain at the project level allows you to invoke separate pass/fail actions based on the pass/fail status of the build.

To set a pass/fail chain at the project level, do the following:

1. Select the name of the project in the **Project** list by clicking the **Edit** icon.
2. In the object edit form (lower content pane), select the Pass Chain field drop-down arrow. Choose the project or library for your desired "Pass" action from the drop-down list.
3. In the Fail Chain field below it, choose the project or library for your desired "Fail" action from the drop-down list.
4. Click **Save** .

Chaining Projects Together

You can link projects together using a feature called chaining. You can use this feature to maintain frequently-used groups of steps separately from projects that depend on them. Other uses include executing automated test and deployment projects on completion of certain steps. Chaining can

Document # SC23-7681-07

also be used to clean up files no longer needed by development teams, by assigning a project to be run on the completion of a job of a specific class.

There are two basic types of chaining:

- Inline: Steps of the called project or library are run inline with the steps of the calling project or library. The properties and environments of the calling project or library are used. Any properties and environments set in the called project or library are ignored, including any chaining set within the called project.

  ### Note

  The server environment of the inlined project's server is used if it is specified.

- Pass Chain or Fail Chain: A pass or fail chain runs conditionally when a step passes or fails. You can specify the same project for both if you want to run it unconditionally.

  A conditionally chained project inherits some characteristics from the calling project but otherwise runs according to its own properties and environments. This behavior distinguishes it from inline chains, which inherit all properties and environments from the calling project.

  - The called project uses its own properties, including its own notification settings and chain settings.
  - The called project applies its own project environments *after* applying the calling project's environments. See also “Environment Variable Inheritance in Chained Projects” on page 35.
  - The called project uses the server specified by its own selector. If it is a library (selector is set to None), it uses the *calling project's server*.
  - The chained project inherits the *class of the calling project* by default. This behavior can be changed in **Administration → System Configuration** by setting Override Class when Chaining to N or n.

If you want to include a platform-independent step as a placeholder step for chaining a project, you can use `.sleep 0` as the step command.

If you use a .break command within a chained project, the system stops the chained project but returns control to the calling project, which then continues. See “.agentupdate” on page 90.

You can create nested sets of chained projects. When you chain (launch) one project from another (using inline chains or pass/fail chains), the chained project can itself be chained to another project. The chain of projects you can create can be no more than 32 levels deep, but is otherwise limited only by the available memory of your Management Console.

Canceling Chained Projects when Wait Enabled

The system cancels projects launched via a Pass or Fail Chain when the calling project or step is canceled, if the Wait check box (on the calling step) is checked. By design, the system normally does not cancel chained projects; however, when the Wait option is used, these projects are

Document # SC23-7681-07

considered more tightly linked to the calling project. The cancellation occurs when the calling project or the calling step within the calling project is canceled.

Launching Projects on Class Events

You can launch (chain) projects when certain events occur that are relevant to classes. Using these properties, you can model a progression of states in your processes.

The following properties of classes allow you to launch jobs when certain events occur:

Start on purge	This property launches the specified project when any job in the class is purged (that is, whenever the system starts a purge job for a job with this class). You can use this property to make sure some specific files are deleted which are not automatically deleted along with the purge.
Start on entry	This property launches the specified project when a job's class property is changed to this class. You can use this property to tie a process to the reclassification of a job; for example, you could create a Test class, and launch some standard tests when a job is promoted to the Test class.
Start on exit	This property launches the specified project when a job's class property is changed from this class to another class.

These properties launch projects as chains.

.run and .runwait

```
.run [-c "<condition>"] "<ProjectName>"
```

```
.runwait [-c "<condition>"] "<ProjectName>"
```

The .run and .runwait commands allow you to launch a chained project from a step command, providing a more flexible means of launching chains, including the evaluation of environment variables to determine whether a chain is launched.

The commands differ in how they behave after they launch a project:

- The .run command launches the specified project as a chain, following the rules for environment variable inheritance for chained projects.
- The .runwait command launches the specified project; then the launching step waits while the launched build completes. When the launched build completes, the system sets the result value of the launching step to the pass or fail status of the launched build (the only values that can result are pass or fail).

Important

A project that includes a step with .runwait consumes two job slots when it runs. If there are insufficient job slots available, the step fails with an error.

Document # SC23-7681-07

Conditional Launches

The optional -c parameter can be used to make the launch depend on a condition. You can use environment variables in the condition. The condition can be of several forms:

String comparison	You can use equal (=) or not equal (!=) operators to evaluate strings. The chain is launched if the comparison evaluates true.
Numeric comparison	You can use <, >, <>, ><, or = operators to compare two numeric values.
Command success	You can use a command enclosed in backticks as the value of the -c parameter. The system runs the command; if it succeeds, the chain is launched.

Example 1. Examples

```
.run "BuildWindowsDriver"
```

The system launches the BuildWindowsDriver project. The launching project continues with the next step immediately.

```
.runwait "BuildWindowsDriver"
```

The system launches the BuildWindowsDriver project. The system pauses the launching project at the .runwait step. When the BuildWindowsDriver project completes and passes, the .runwait step's status is set to pass.

```
.run -c "$HOMEDRIVE=C:" "Simple Echo"
```

The system runs the project Simple Echo if and only if the HOMEDRIVE variable has the value C:.

This command produces log output like the following (in the EXEC section of the step log):

- When HOMEDRIVE is C:

```
.run Condition: 'C:' = 'C:' satisfied.

Queueing Project "Simple Echo" on server [WinBox].
Queued Build 'BUILD_202' of project 'Simple Echo'.
```

- When HOMEDRIVE is not C:

```
.run -c "$HOMEDRIVE=C:" "Simple Echo"

.run Condition: 'D:' = 'C:' unsatisfied, no project queued.
```

The system can numerically compare strings if they contain numbers. For example, it handles the following cases as shown:

.runwait -c "a12b<c42d" "Simple Echo"	`.run Condition: '12' < '42' satisfied.`
	`Queueing Project "Simple Echo" on server [WinBox].`
	`Waiting for .run build (4411) to complete.`
	`.run build is now running.`

Document # SC23-7681-07

```
                                .run build has finished.
                                Build 'BUILD_203' of project 'Simple Echo' completed.
```

.runwait -c "f43g<>h43i" "Simple Echo"

```
.run Condition: '43' <> '43' unsatisfied, no project
queued.
```

The following examples show how to use commands as conditions. Note that the command must be enclosed in both quotes and backticks:

.run -c "`exit 1`" "Simple Echo"

```
Env .run encountered an error during variable expansion,
parameter [`exit1`] expanded to [].
Expansion returned non-zero exit, project will not be
queued.
```

.run -c "`exit 0`" "Simple Echo"

```
Expansion returned zero exit, project will be queued.
Queueing Project "Simple Echo" on server [WinBox].
Queued Build 'BUILD_204' of project 'Simple Echo'.
```

When you use .runwait and a build fails, the log looks like the following:

.runwait "Fail Build"

```
Queueing Project "Fail Build" on server [WinBox].
Waiting for .run build (4413) to complete.
.run build is now running.
.run build has finished.
Build 'BUILD_3' of project 'Fail Build' Failed, setting step
 status to fail.
```

Using Libraries

From the point of view of the Management Console, a library project is any project whose Selector property is set to None. These projects are intended to be run within other projects and therefore use the server of the step that calls them.

When you attempt to save a project with a selector of None, the system warns you that it will become a library project. Library projects are not listed in the **Projects** module; they appear in the **Libraries** module.

To use a library project, link it to a step in your project by selecting it as the Inline, Pass Chain, or Fail Chain project for the step.

About Libraries

The Libraries module displays library projects, which do not have a selector specified. These projects absorb the selector of any project which calls them; they are typically called by other projects via inline chains or pass/fail chains.

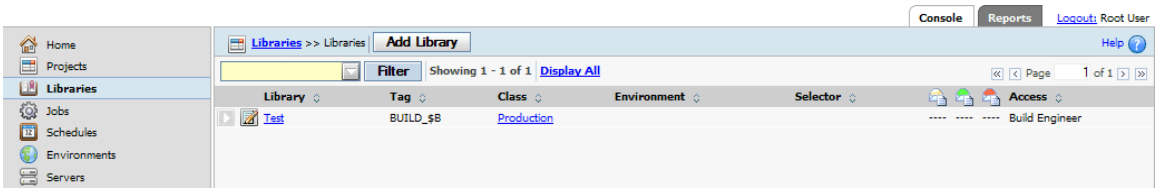
From the Libraries module, you can view, edit, create, or launch library projects. You can execute a library project by itself, but you must specify a selector when you do so.

Document # SC23-7681-07

You can change a library project into a normal project by editing the project and choosing a selector for it. When you save a library project with a selector, it becomes an ordinary project, disappearing from the Libraries list.

Aside from the lack of a selector, libraries are treated just like any other project.

The Libraries Module



Copying a Library

To make a copy of an existing project, display the existing project's list of steps by selecting **Projects** → **<ProjectName>** , then click the **Copy Project** button.

Select similar options from the **Library** module to make a copy of a library.

The system displays a dialog box to prompt you for a name for the new project or library.

When you copy a project or library, the system copies the following aspects of the existing object to the new object:

- The steps and all of their properties
- All the project properties listed on the **Project Details** tab, such as the project's tag format, class, selector, and other properties.

The system does not copy the following properties:

- Tag variables (found in the **Tags** tab on the project properties form)
- Project registers (found in the **Registers** tab on the project properties form)

Deleting/Clobbering Projects

There are two ways to delete projects, depending on whether the project has any jobs associated with it.

Choose one of the following options for deleting projects:

Delete Project button

The **Delete Project** button can delete projects which have no jobs. The button appears on the project property editing page and the project step list. To view project-level properties, select **Projects** , then click the icon next to the desired project's name. Deleting a project cannot be undone. If you want to

Document # SC23-7681-07

	delete a project with this button, make sure all of the project's jobs are deleted first.
Clobber button	The **Clobber** button deletes a project and all of its associated jobs from the Build Forge database. The system asks for confirmation before clobbering a project. Clobbering a project cannot be undone. The button appears on the project property editing page; select **Projects** , then click the  icon next to the desired project's name.

Log Filters

Use log filters to change the success criteria for a step. A filter stores one or several regular expressions; when it finds one of its expressions in step output, it changes the outcome of the step.

You can create new filters, add criteria to them, and edit existing criteria from the **Projects → Log Filters** pages.

Creating a Log Filter

Log filters may contain one or more filter patterns. Each filter pattern is associated with an action, and optionally an access group for notification. You define a log filter first and then associate the log filter with a step in the project.

To create a log filter, do the following:

1. Select **Projects → Log Filters** .

   The Management Console displays the list of Log Filters and the New Log Filter form.

2. At **Name** , enter a name for the log filter, then click **Save** .

   The Management Console saves the log filter and displays the New Pattern form.

3. For each filter pattern that you define for the log filter, do the following:

   a. Enter a regular expression in the **Pattern** field. (The regular expression must be Perl-compatible.) Build Forge searches step output for the pattern when the project runs. For details, see “Filter Patterns” on page 80.

   b. At **Action** , choose a filter action to take when the filter pattern is found. The default property, Set Fail, sets the step status to Fail. For details, see Filter Actions.

   c. At Notify, optionally select an access group to send members a notification e-mail when the filter is activated.

   d. Click **Save** .

Document # SC23-7681-07

To use the log filter, choose a project step and set the step's **Result** property to the new log filter. See Assigning a Log Filter to a Step.

Filter Actions

Filter actions define what action is taken when a filter pattern is found in step output. Each filter pattern you create is associated with a single filter action. Both filter actions and patterns are defined in log filters.

To create a log filter, select **Projects** → **Log Filters** . For details, see Creating a Log Filter.

Filter Action	Definition	Step Results
Set Fail (the default)	When the system finds the filter pattern, it sets the step results status to Fail and continues searching the current line for filter patterns in the set.	Fail
Set Fail/Halt	When the system finds the filter pattern, it sets the step results status to Fail, stops searching the current line for the filter patterns in the set, skips to the next line, and starts the pattern search again.	Fail
Clear Fail	When the system finds the filter pattern, it sets the step results status to Pass and continues searching the current line for the filter patterns in the set.	Pass
Clear Fail/Halt	When the system finds the filter pattern, it sets the step results status to Pass, stops searching the current line for the filter patterns in the set, skips to the next line, and starts the pattern search again.	Pass
Halt	When the system finds the filter pattern, it stops searching the current line for the filter patterns in the set, skips to the next line, and starts the pattern search again. It does not change the step results status.	not applicable
Include	Include allows you to reference one or more log filters in another log filter. You specify the log filter you want to include in the Pattern field and select Include in the Action field.	not applicable
Warning	When the system finds the filter pattern, it sets the step results status to Warn and continues searching the current line for the filter patterns in the set.	Warning
Clear Warning	When the system finds the filter pattern, it sets the step results status to Pass and continues searching the current line for filter patterns in the set.	Pass
Clear Warning/Halt	When the system finds the filter pattern, it sets the step results status to Pass, stops searching the current line for the filter patterns in the set, skips to the next line, and starts the pattern search again.	Pass

Document # SC23-7681-07

Filter Action	Definition	Step Results
Notify Changers	To use Notify Changers, an adaptor that creates a relationship list must be included in the project and the adaptor step must run before the step that contains the Notify Changers log filter. The adaptor relationship list pairs users and objects (such as changed files). For details, see the Adaptor XML Reference. After the adaptor runs and creates the relationship list, if a log filter with the Notify Changers action matches its filter pattern in a step output line, the line is scanned again to try to match objects in the relationship list. If an object match is found, the users paired with the object are sent e-mail notification. For example in the following line of step output, the object match for the filter pattern `Error` is MyFile.c. So, the users paired with the MyFile.c object in the relationship list are sent e-mail notification of the error. `Error: Invalid token on line 55 of MyFile.c`	not applicable

Error Thresholds

You can use a special environment variable, _ERROR_THRESHOLD, to establish thresholds for individual steps and/or for a project. The system then counts the number of filter matches, and either fails the step or project when the threshold value is met, or notes the fact that the threshold was met in the job notes.

For more information, see “Special Case Variables” on page 40.

Error and Warning Counts

If filters are associated with steps to determine whether the steps succeed or fail, the system displays the number of errors and warnings caught by the filters. In the **Jobs → Completed**) tab, the numbers appear in in the **Results** column in parentheses after the job result. The format is **(<** *fail matches* **> / <** *warning matches* **>)**.

Example: A job result of **Failed (1 / 0)** shows that the job failed, 1 Fail filter was matched, and no Warning filters were matched.

Filter Patterns

A filter pattern defines the character string or expression that you want to match in step output. Each filter pattern you create is associated with a single filter action. Both filter patterns and actions are defined in filter log sets. The ability to include multiple filter patterns in a log filter and apply it to output from a single step allows you to use multiple search criteria without constructing complex expressions.

To create a log filter, select **Projects → Log Filters** . For details, see Creating a Log Filter.

Filter Pattern Syntax

Review these guidelines for creating filter patterns:

- The filter pattern is defined as a regular expression and must use Perl-compatible syntax. For details about constructing Perl-compatible expressions, refer to Perl documentation.
- The system adds the delimiting forward slash characters (/<expression>/), so specify the expression **without** surrounding forward slash delimiters (expression).
- If your expression includes a metacharacter (for example, a /b), the metacharacter must be preceded by a backslash escape character (a\/b).

Syntax for some standard regular expressions are shown in the following table.

Expression	Matches
Production	Matches the *Production* anywhere in the string.
^Production	Matches *Production* at the beginning of the string.
Error:.*[0-9]$	Matches a line that contains *Error* followed by any set of characters terminated by a number at the end of the string.
[Ww]arning	Matches *Warning* or *warning*.
.*	Matches 0 or more of any character. The dot (.) matches any character, and the asterisk (*) matches any character 0 or more times.

Multiple Pattern Matches on the Same Line

To construct a pattern filter, it is important to understand how the system searches for pattern matches.

For each line of output, the system checks for matches against all the filter patterns in order; it stops when it finds a match, and moves on to the next pattern. So, if the pattern occurs twice on one line, the system may not find it.

```
exception retrying exception
```

For example, using the previous line of step output and the filter patterns in the following table, the system would match the first *exception*, set the step result to Fail, match *retrying* and set the step result to Pass, and move on to the next line without matching the second *exception*.

One way to resolve this problem is to replace the filter patterns in the table with the following filter pattern:

```
retrying.*exception
```

Filter Patterns	Filter Actions	Example Description
[Ee]xception [Rr]etrying	Set Fail - Fail Clear Fail - Pass	This is useful for Java projects; it fails the step on exceptions, but clears the failure on a retry. If the retry fails, a new exception will be generated, so that the final state of the command is valid.

Document # SC23-7681-07

Using Classes

A class is a group of jobs; each job must be a member of one and only one class. You can use classes to set up types of jobs, and apply behavior to each type globally. A job gets its default class from the properties of its project, but you can manually choose a different class for a job when you launch it from the **Jobs → Start** page.

Classes have properties to manage the following activities:

- Deleting jobs automatically
- Launching jobs when the system purges job of this class, or when an existing job is changed to or from this class.

Note

You can change the class of a job after it completes. To change the class of a job, view the job by selecting **Jobs → Completed** and then click the job tag (BUILD_5, for example). Select a different class in the **Class** field.

To edit the properties of a class, select the **Projects → Classes** page. The system displays a list of classes; click an existing class, or click the **Add Class** button.

The **Access** property of a class controls which users can view or change it, based on the access group you assign.

About Classes

A class is a group of jobs; each job must be a member of one and only one class. You can use classes to set up types of jobs, and apply behavior to each type globally. A job gets its default class from the properties of its project, but you can manually choose a different class for a job when you launch it from the **Jobs → Start** page.

Classes have properties to manage the following activities:

- Deleting jobs automatically
- Launching jobs when the system purges job of this class, or when an existing job is changed to or from this class.

Note

You can change the class of a job after it completes. To change the class of a job, view the job by selecting **Jobs → Completed** and then click the job tag (BUILD_5, for example). Select a different class in the **Class** field.

To edit the properties of a class, select the **Projects → Classes** page. The system displays a list of classes; click an existing class, or click the **Add Class** button.

The **Access** property of a class controls which users can view or change it, based on the access group you assign.

Document # SC23-7681-07

Deletion Properties of Classes

Most of the properties for classes control what kinds of project data are deleted and under what conditions they are deleted.

The system checks for jobs to delete at an interval defined by the Purge Check Time system setting, which defaults to 15 minutes.

Note

You can also use schedules to denote when purges should be performed, so that the system does not try to run purges when the system is otherwise occupied. You can use this feature to have purges occur only at night, or once a week, for example. See “Scheduling Purges” on page 128.

Delete Files: Determines what kinds of data are deleted. It has the following options:

Everything — Deletes all information about the job from the database and deletes the job directory from the server(s) that ran it.

Console Data — Deletes all information about the job from the database, but leaves the job directory on the server intact.

Logs And Files — Deletes the job directory and the logs, but retains step pass/failure information on the **Jobs → Archived** page.

Logs Only — Deletes only the job logs.

Files Only — Deletes the job directory on the server(s) that ran the job. Logs and some other information (such as step pass/fail status) remain within the database; the job record moves to the **Jobs → Archived** page.

Days: The number of days old a job must be before it is deleted.

Count: The maximum number of jobs allowed. When the number of jobs exceeds the Count value, the system schedules purge jobs to delete the extra builds. The default value, Unlimited, prevents the system from deleting jobs because of the number of jobs that exist.

Note: The system deletes jobs when **either** the Days or Count values are exceeded. For example, if you have Count set to 10 and Days set to 2, and there are 8 jobs, but 3 are more than 2 days old, those three jobs would be deleted. Similarly, if you had 12 jobs, all less than 2 days old, the two oldest jobs would be deleted.

Which: The Which property sets additional conditions that must be met before a job can be deleted. It has the following options:

Any Build — When this option is selected, the Which property has no effect on job deletion.

Only Failed — The system deletes only failed jobs.

Only Passed — The system deletes only passed jobs.

Document # SC23-7681-07

Keep 1 Pass — The system always retains the most recent passed job, even if it meets other deletion criteria.

Launching Projects on Class Events

You can launch (chain) projects when certain events occur that are relevant to classes. Using these properties, you can model a progression of states in your processes.

The following properties of classes allow you to launch jobs when certain events occur:

Start on purge — This property launches the specified project when any job in the class is purged (that is, whenever the system starts a purge job for a job with this class). You can use this property to make sure some specific files are deleted which are not automatically deleted along with the purge.

Start on entry — This property launches the specified project when a job's class property is changed to this class. You can use this property to tie a process to the reclassification of a job; for example, you could create a Test class, and launch some standard tests when a job is promoted to the Test class.

Start on exit — This property launches the specified project when a job's class property is changed from this class to another class.

These properties launch projects as chains.

Setting Up Notification

The system can send out e-mail notifications when projects or steps pass or fail, or when certain other events occur. This section describes how to configure e-mail notifications and how to modify the notification templates that control what e-mail notifications look like.

Notifications are sent to access groups, so design your access groups with notifications as well as security in mind.

Note

Notifications are always sent to groups, not individual users directly, but you can set up groups that contain only one user, if needed.

To create a notification event, select an access group for a notification property.

- For projects, you can choose Start Notify, Pass Notify, and Fail Notify groups.
- For steps, you can choose Pass Notify and Fail Notify groups.

Whenever these properties have access groups selected, the system sends e-mail to the group members when the appropriate event occurs.

When a project includes another project as an inline project, the inlined project's start, pass, and fail notification settings are ignored, but any notification settings for its steps are honored. See “Notification for Inlined Projects” on page 68 for details.

Document # SC23-7681-07

Before you can use notification, you must:

- Configure the SMTP Server system setting so that the system knows what SMTP server to use to send e-mail. The default is localhost. You might also need to set the system setting for **System Alert Source** . This address is used as the source address and most SMTP servers require a valid source address. The default is root@localhost.
- Create one or more notification groups and assign users to them.
- Select groups to notify for individual projects and/or steps.

In addition, you can configure the notification e-mails that the system sends out by editing notification templates. See About Notification Templates.

About Notification Templates

Notification templates provide a means of sending customized messages to user about events in the system. The system includes a number of templates which you can customize for your organization's needs. You can also create templates for specific projects using the **Projects → Templates** page. Plain text templates are sent as plain text e-mail messages; templates that contain common HTML markup are sent as MIME messages. The system parses template bodies for a number of variables (see “Using Environment Variables and Register Variables in Templates” on page 67).

Configuring Your SMTP Server

To configure your SMTP server, select **Administration → System → SMTP Server** . The system displays an edit form for the SMTP Server value. Enter the name of your site's SMTP Server. The default is localhost.

You might also need to set the **System Alert Source** parameter as this is used as the source address and most SMTP servers require a valid source address. The default is root@localhost.

For more information, see “System Configuration Settings” on page 164.

Setting Notification Properties of Projects and Steps

When the SMTP and group configuration for notification is in place, you can configure projects and steps to send notifications when certain events occur.

- For projects, you can set Start, Pass, and Fail Notify properties. These are project properties.
- For steps, you can set Pass or Fail Notify properties. These are step properties.
- By default, the **From:** field populates as "Build Forge Management Console." To meet your needs, you can enter a different value, which the group members will see in the **From:** field of the e-mail notification they receive.

Document # SC23-7681-07

Notification Exercise

The following procedure describes how to set up and try out e-mail notification. The exercise requires an SMTP server and an e-mail account.

1. Set up your SMTP server as described in “Configuring Your SMTP Server” on page 65

   Make sure you have a user account that sends e-mail to an e-mail account you can access.

2. Select **Administration → Access Groups**

3. Create a new access group called Email Test with your chosen user as the Initial Member.

4. Select a project (for example, the Hello World project) and edit its project properties. Select the Email Test group in the Start Notify, Pass Notify, and Fail Notify fields.

5. Run the project.

6. Verify that you received two e-mails: one to indicate the start of the project, and one to indicate its success or failure. If you do not receive the e-mails, verify that you used the correct SMTP server value.

Customizing Notification Templates

Notification templates provide a means of sending customized messages to user about events in the system. The system includes a number of templates which you can customize for your organization's needs. You can also create templates for specific projects using the **Projects → Templates** page. Plain text templates are sent as plain text e-mail messages; templates that contain common HTML markup are sent as MIME messages. The system parses template bodies for a number of variables (see “Using Environment Variables and Register Variables in Templates” on page 67).

Creating New Templates for Specific Projects and Steps

The system comes with templates for many events that can occur in the system. You can create new ones that are specific to a particular project and/or step. When you do this, the more specific template is used instead of the global project template. For example, you can create a variant of the normal Project Run Start message that applies only to your FinalProductionWidget project.

To create a new template, select **Projects → Templates** . The system displays the current list of templates. Click **Add Template** to add a new one.

When you click **Add Template** , the system offers you a sequence of choices. You can select a project, step, and type of notification:

- Project: Choose the project that the new template applies to. The template is only used on notification messages that are generated for runs of the selected project.
- Step: You can choose a specific step (so that the template only applies to notifications for that step) or select Project Events to have the template apply to all notifications for the selected project.

Document # SC23-7681-07

- Notification: If you selected Project Events as the Step option, you can choose from a list of project events. If you chose a specific step, you can choose from Step Pass, Step Warn, and Step fail messages for your notification type.
- From: By default, the From: field automatically populates as "Build Forge Management Console." To meet your needs, you can enter a different value, which the group members will see in the From: field of the e-mail notification they receive.

Editing Notification Templates

To edit a notification template, select **Projects → Templates** , then click on the name of the template you want to edit. Newly-created templates default to the same text as standard templates, until you edit them.

Using Environment Variables and Register Variables in Templates

You can reference environment variables (ones defined by you as well as standard system variables) in notification templates, so long as you use the ${VAR} syntax.

You can also include register variables for a project in notification templates. If you reference an empty register, the system returns an empty string.

Special Notification Template Variables

The following table lists the special variables available to notification templates. Some variables are context sensitive and are only available when relevant (for example, the STEPNAME variable is not set for project level notifications, only step level notifications).

Variable	Contains
ACTION	For purge templates, describes the type of deletion performed.
BID	Specifies the job ID number. Used to construct links back to the Management Console to access reports.
CMD	For source adaptors, identifies link command specific error strings.
CONSOLEHOST	The host name of the Management Console machine.
CONSOLEPORT	The port number used by the Management Console. Allows you to construct valid URLs within a notification template.
CONTEXTLOGLINKS	Lists lines from the log that begin with “FILT:”, with three lines of context per entry. The system provides links to the Management Console log entries in the message.
DURATION	For steps, specifies the number of seconds the step ran.
EID	Specifies the Environment ID number. Used to construct links back to the Management Console to access reports.
FULLNORMALLOG	Shows the log information for each step in the job, excluding the environment setup actions that appear in the detailed log.
LINK	For source adaptors, specifies the link name.
MESSAGE	Contains the error or message text for failure or alert messages.

Document # SC23-7681-07

Variable	Contains
ONFAIL	For steps, holds the continue property for the step.
PATH	Specifies paths where appropriate, for data items like servers or steps.
PID	Specifies the Project ID number. Used to construct links back to the Management Console to access reports.
PROJECTNAME	Contains the name of the project.
SERVER	Contains the logical server name for a step.
SID	Specifies the Step ID number. Used to construct links back to the Management Console to access reports.
SRVRHOST	Contains the TCP/IP host name of the server for a step.
START	Contains the date/time a job started.
STEPNAME	For steps, contains the name of the step.
STEPNORMALLOG	Shows the log information for the current step in the job, excluding the environment setup actions that appear in the detailed log.
TAG	Contains the tag string for a job. The same value as $BF_TAG.
TAILNORMALLOG	Works like FULLNORMALLOG, but only displays the end of the log. The number of lines displayed is controlled by the **Tail Log Amount for Mail Template** system setting. This variable must be used in a step notification template.
UID	Specifies the User ID number. Used to construct links back to the Management Console to access reports.
USEREMAIL	Contains the e-mail address for the owner of a job/event.
USERNAME	Contains the full name for the owner of a job/event.

Notification for Inlined Projects

The system handles notifications for inlined projects as if the steps from the inlined project were embedded in the calling project:

- When a project contains an inlined project, the inlined project's project-level notification settings are ignored. When Project A inlines Project B, no messages about the start, passing, or failure of Project B are sent.
- Step-level notification settings are unaffected. If a step has a Pass or Fail Notify access group set, the appropriate messages get sent, whether the step is in a top-level project or an inlined project.
- The inlined steps contribute to determining whether the calling project succeeds or fails. A failure in an inlined step, for example, either causes the calling project to fail or, if the step is set to Continue On Failure, changes the calling project's state to Passed with Warnings.

Document # SC23-7681-07

Working with Steps

This section describes how to create and manage steps in the Management Console.

About Steps

A step is a component of a project. When the project is run as a job, each step is executed in order. A step contains one or more commands and has step properties that affect its behavior.

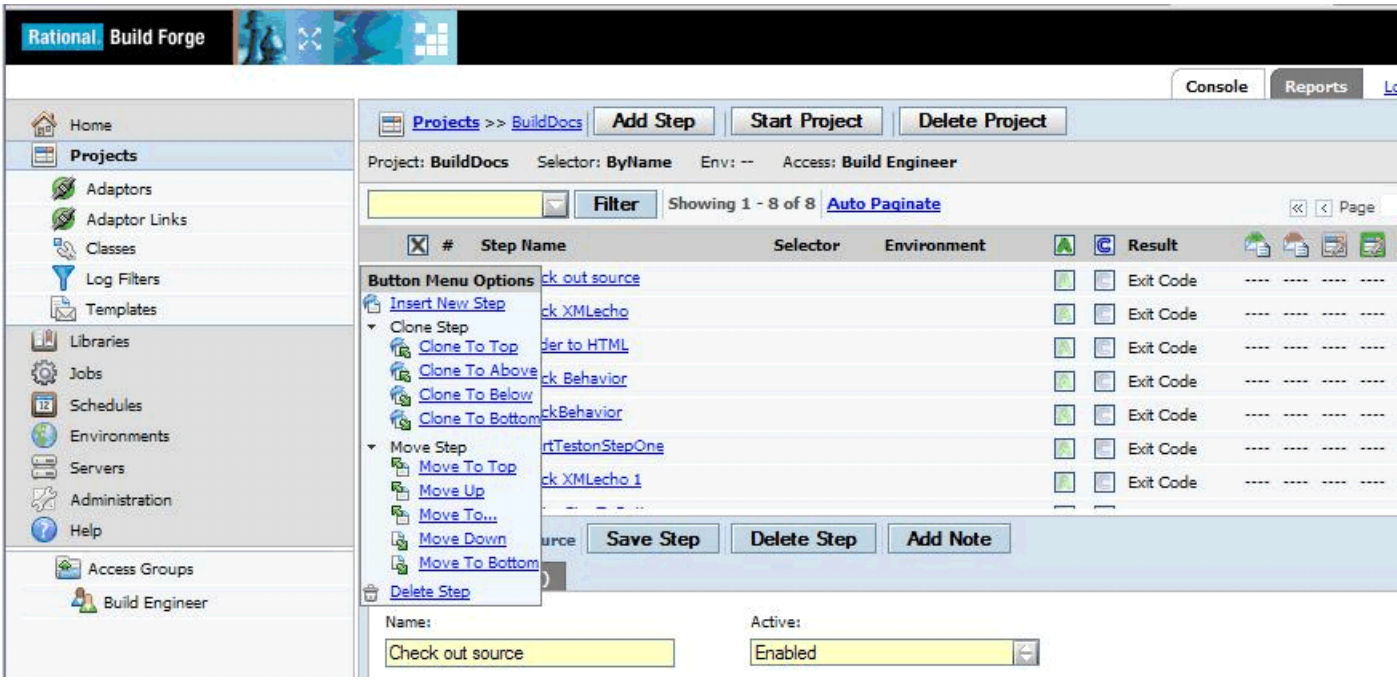
To work with steps, select **Projects** , then click on a project name. The system displays the list of steps for the selected project. When the list is displayed you can add, edit, copy, and move steps.

- **Editing a step**: click the step name. The system displays the step's properties in the lower pane. Make your changes, then click the **Save Step** button.
- **Disabling a step**: click the check box in front of the step name. When a step is checked  , it has the following effects on the project:
  - The step is not executed when the project is run as a job.
  - The step is greyed out in the list of steps when you start a job normally: select **Jobs** → **Start** , then click the job name, then click **Job Steps** . The step is visible but cannot be made active for the job you are starting.

  A step can also be disabled by setting the **Active** property to Disabled in the **Details** tab for the step. When you save the step, the check box is checked.
- **Adding a step to the end of the project**: click the **Add Step** button at the top of the main pane. The system displays an empty step details form. Enter values for the properties (you must enter a **Name** and **Command** at least), then click the **Save Step** button.
- **Other step operations**: click the  Actions icon in front of the step name to display additional options:
  - **Insert New Step** :
  - **Clone Step** : copy a step and all of its properties. The name is changed to add a number at the end of the step. Copying a step named Build results in a new step named Build 1. The number is set automatically. You can copy to these locations:
    - Top: start of the list
    - Above: immediately before the current step
    - Below: immediately after the current step
    - Bottom: bottom of the list
  - **Move Step** : move a step to a different position in the list. You can move to these locations:

Document # SC23-7681-07

- Top: start of the list
- Up: move up one position
- Move to...: a dialog requests the step number where specify a step number. The step is moved to that position and other step numbers are adjusted as needed.
- Down: move down one position
- Bottom: bottom of the list



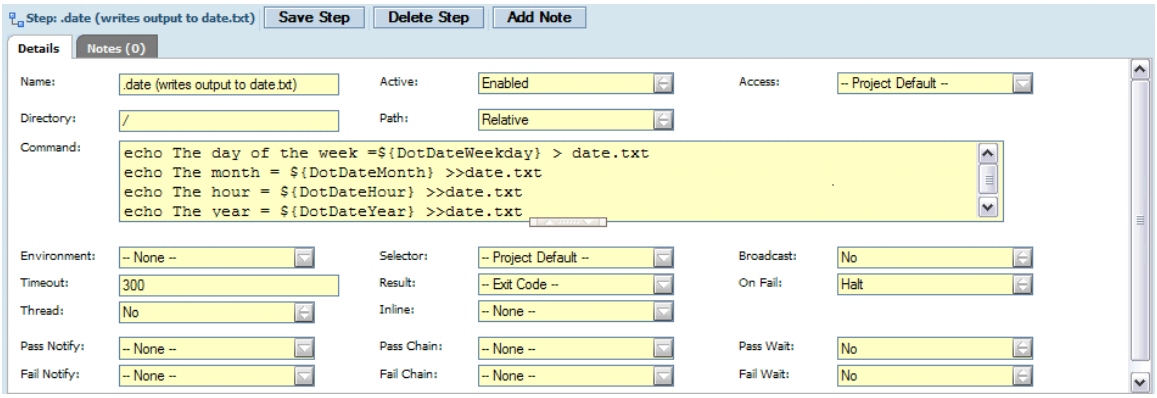
Step Properties

Step properties specify how to execute a step, handle its output, and what to do upon completion. A step command can also execute another project or library.

If a step property is not set explicitly, its value is inherited from the project.

Document # SC23-7681-07

Step Details Form, Showing Step Properties



Step properties include:

Name — The name of the step; a label for the step within the system.

Active — By default, all steps are enabled. To disable a step, select Disabled. A disabled step is not used when the system runs the project; it does not appear in the list of available steps if a user manually starts the project and selects specific steps to include or exclude. Use this feature to prevent an unfinished step from being executed, or to temporarily remove a problem step from a project.

Access — Choose an Access Group to define which users are allowed to use the step. You can use this property to restrict access to specific steps within a project. When a user who is not a member of the Access Group for a step launches the project that contains the step, the step is skipped.

Choosing Project Default causes the step to inherit the access properties of the project.

Directory and Path — The **Directory** field sets the final portion of the path for step command execution. By default, the **Path** is relative to the project path created on the server. The system automatically creates a unique directory for every job, and most steps should execute there. The **Directory** field provides a convenient way to execute commands in directories your project has constructed during a job. (Build Forge does not construct directories mentioned in the **Directory** field.)
The Absolute option for the **Path** field allows a step to ignore the project and job directories and access directories relative to the server's **Path** field.

- If **Relative** is selected for the **Path** field, then step commands are executed in a path found by adding together the server, project, job, and step directories.
- If **Absolute** is selected for the **Path** field, then step commands are executed in a path found by adding together the server and step

Document # SC23-7681-07

directories. This option allows you to launch applications which are permanently installed on a server, separate from the project directory structure.

Command

Enter one or more operating system commands, subject to the following guidelines:

- Separate individual commands by placing them on separate lines.
- All the commands in a single step are executed on one server. If you want to be able to thread commands across multiple servers, you must place those commands in separate steps.
- If you use more than one command in step, you may want to use command filters as the step's **Result** property. When you use the default Exit Code setting for the **Result** property of your step, the success or failure of the entire step is determined by the last command in the step. Earlier commands may succeed or fail, but the system only looks at the exit code of the last command. When you use a filter, you can catch failures in earlier commands.
- You can enter multiple commands.
- You can use the #! directive to specify the shell to be used to execute the command(s). This works on Windows® as well as Linux® and UNIX® systems (the Windows agent handles passing the commands to the specified interpreter). To send the commands from your step to a copy of Perl in c:\perl\bin on Windows, use #!c:\perl\bin\perl.exe If you use the Windows agent with Cygwin, but need to direct a command to the Windows shell cmd.exe, you can use the following line, which takes advantage of Windows implicit paths:

  ```
  #!cmd.exe /C
  ```

  Note that the /C option is required for cmd.exe as otherwise it waits for additional commands after your step commands are delivered to it. You might use the `#!/bin/perl` command on a UNIX or Linux machine.

Note

When you use the `#!` command on Linux or UNIX systems, the system does not change to the standard default directory (the path constructed from a combination of the server path, project, name, and step path field) because it cannot predict the required syntax; you must include your own directory-changing command. Use special environment variables created by the system, such as BF_SERVER_ROOT and BF_PROJECTNAME_PHYS, to do this.

Document # SC23-7681-07

Environment	Environment: Choose an environment. The system applies the values in the environment you select before it runs the command. The final environment depends on inheritance from the server and project variables as well as the step variables. Choosing Project Default causes the step to inherit the environment of the project.
Selector	The name of the selector to use when choosing a server for this project. You can leave this field blank to use the project's selector.
Broadcast	Check this box to run the step on **every** server that matches the step's selector. At run time, the system replaces a broadcast step with a series of steps, one for each server, and executes them serially or in parallel, depending on the broadcast step's **Thread** property.
Timeout	Specifies how long the system should wait for the current command to produce output. If the timeout value is reached, the system fails the project at the current step (unless the step is set to continue on fail).
Result	The Result property determines how the system judges whether a step succeeded or failed. Use the default value of Exit Code to determine success based on an exit code returned by the command shell. Or, choose a filter which examines the command output to determine success.
On Fail	The On Fail property determines whether the system Halts or Continues with the following step if the step fails. By default, the system halts and does not continue if the step fails.
Thread	Set this property to Yes to allow threading of this step (running the step in parallel with other steps). Set the property to No to avoid threading. Set the property to Join to separate threaded blocks of steps. The first set of steps must complete before the next set of threaded steps following the Join step can start.
Inline	Specifies a project to run inline with the current project (an inline chain), after the current step.
Pass Notify	This property specifies an access group to be notified if the step passes.
Pass Chain	Specifies a project to launch on the success of the current step. The system displays the chained steps as indented steps in the Jobs reports.
Pass Wait	If this box is checked, the system suspends the current project until the pass chained project completes.
Fail Notify	Specifies an access group to be notified if the step fails.
Fail Chain	Specifies a project to launch on the failure of the current step. The system displays the chained steps as indented steps in the Jobs reports.

Document # SC23-7681-07

Fail Wait	If this box is checked, the system suspends the current project until the fail chained project completes.

Labelling Log Output

Use a prefix in front of output lines to have output listed in a labeled category in the step log.

This task assumes that you have already created a selector, server, and project. It assumes that you are using a server with an operating system that accepts the echo command (for example, Windows, Linux, or UNIX).

You can include an uppercase label at the beginning of any line of output. When a line begins with a label, Build Forge repeats the output in the category with the same name as the label, or creates a new category if the category does not already exist. The system recognizes a label when it follows these rules:

- The label must be composed solely of uppercase letters (no spaces or punctuation) and followed by a colon. SPACESHIPS: is valid; Space Ships: is not.
- The label must be at least 3 letters long.

After a line that begins with a label, subsequent output lines are copied to the same category until you use a new label. Use the procedure below to try out this feature.

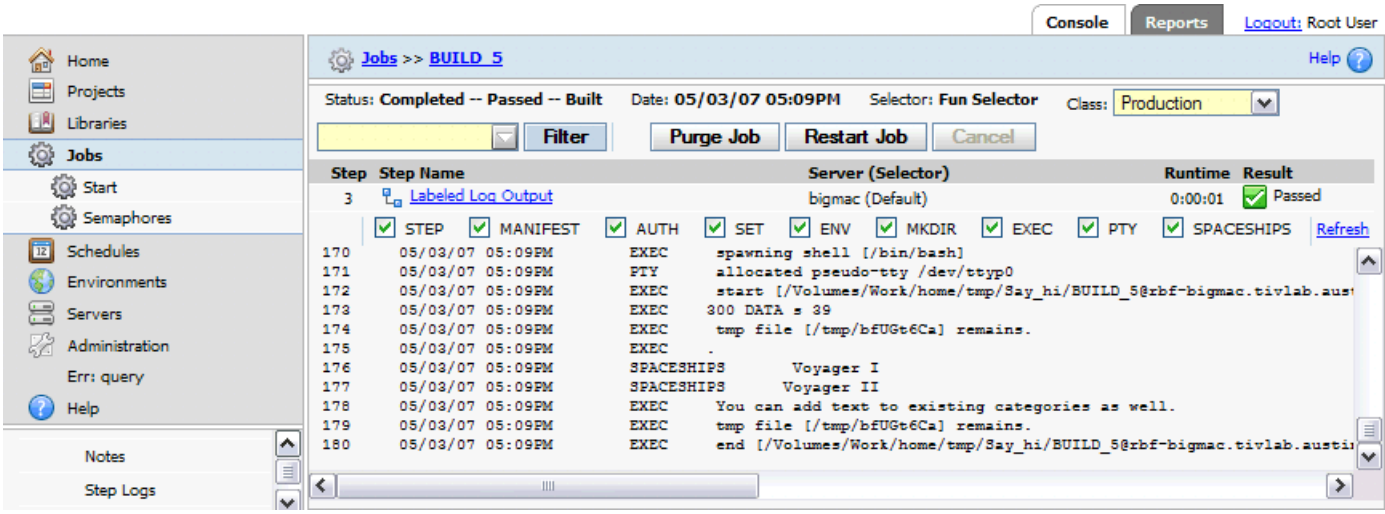
1. Create a new step in one of your projects. Give the new step a name like LabeledLogOutput.
2. For the step's command, enter the following text.

```
echo SPACESHIPS: Voyager I
echo Voyager II
echo EXEC: (You can add text to existing categories as well)
```

3. Run the project. When it completes, view the log for the new step. You should see output similar to the following.

Document # SC23-7681-07

Labeled Log Output



Note

You have to click on the new SPACESHIPS category to expand its contents. The system displays the category name as a Show option; you can select or deselect the SPACESHIPS check box to show or hide the category.

Whenever you prefix an output line with an all-caps name and a colon, the system copies the output to a labeled category.

Creating Links and Highlighting in Step Logs

The Management Console log viewer recognizes two special bracket commands in step output. When these commands appear in step output, the system displays the output in special ways:

- Surround text with the [STATUS][/STATUS] wrapper to highlight it in the logs.
- Use the [URL][/URL] wrapper to make a URL into an active hyperlink.

[STATUS] Examples

In a step, you can use a command like the following to highlight output from a command. The [STATUS][/STATUS] wrapper marks the wrapped text:

```
echo [STATUS=WARN]WARNING DANGER WILL ROBINSON[/STATUS]
```

You can use any of the following variations:

Command	Highlight Color
[STATUS=Pass]	Green
[STATUS=Warn]	Yellow
[STATUS=Fail]	Red

Document # SC23-7681-07

Command	Highlight Color
[STATUS=Run]	Blue

NOTE: The start [STATUS] and end [/STATUS] commands must be on the same line of the output.

[URL] Example

Use the following syntax in command output to create an active hyperlink in your logs:

```
[URL]http://www.ibm.com[/URL]
```

You can use this syntax in an echo command or any command that produces output. The system displays the text as an active hyperlink to the indicated URL. The start and end tags must appear on the same line of the output.

These commands need not be upper-case; [url][/url] and [status][/status] work as well.

You can include the [STATUS] tag in your regular expressions by adding the "\" escape character to identify the tag inserted in the line.

For example, to find the word "Access" at the beginning of a line where there is a STATUS=RUN tag, enter:

```
^\[STATUS=RUN\]Access
```

Adding Notes to Steps

When you view a step, you can enter notes about it via a **Notes** tab. These notes can serve as an audit trail or change history for the project. In addition to appearing in the project record, the notes (current to the time of the job) appear in the BOM for the job.

Each entry becomes a new note, with a date and user stamp. Individual notes are displayed in a table, and can be deleted via a trash can icon.

To add a note, select a step, then click the **Notes** tab. Click **Add Note** , then enter your note text and click **Submit** to save the note.

To edit a note, click the  icon to display the note text in the Note field. Edit the text and click **Submit** .

Launching Projects from Steps

The system provides several different methods for launching a project from a step within another project.

To launch a project from a step, use one of the following options:

- Specify the project you want to launch as the Inline property of a step. An inlined project appears in the log of the calling project, and tends to inherit environment variables from its caller.

Document # SC23-7681-07

- Specify the project you want to launch as the Pass Chain or Fail Chain property of a step. An chained project is launched as a separate job, and does not tend to inherit environment variables from its caller.
- Use the .run or .runwait command in the step's command field. A project launched this way acts like a chain.

Broadcasting a Step to Multiple Servers

When you have an activity that can be usefully performed on many servers, you can use the broadcast feature to repeat the same step on many servers.

Normally, a step runs on only one server. However, each step has a **Broadcast** check box. When a step's **Broadcast** box is checked, at run time the system replaces the step with a set of non-broadcast steps, one for each server that matches the step's selector.

Note

If the selector for the step matches only one server, then the step runs only once.

Potential uses for broadcasting include:

- Rebooting a group of servers.
- Running a test on a group of servers.
- Copying the same set of files to a whole group of servers.
- Checking out the same set of source code to multiple servers, preparing them for later individual tasks with a single, easy-to-maintain step.

Threading in Broadcast Steps

When it creates replacement steps for a broadcast step at run time, the system threads steps as follows:

- If the broadcast step's **Thread** property is set to **No** , the replacement steps get the same **Thread** value, and thus all execute in series. Each step must complete before the next one can start.
- If the broadcast step's **Thread** property is set to **Yes** , the replacement steps also get the same **Thread** values. This results in a set of steps which execute in parallel with each other and with any threaded steps which precede or follow the broadcast step.
- If the broadcast step's **Thread** property is set to **Join** , the system creates the replacement steps with **Thread** set to **Yes** , except the last step, which is marked **Join** . The result is a set of steps which execute in parallel with each other, and with any threaded steps which precede them, but the whole set must complete before the step following the broadcast step can start.

Document # SC23-7681-07

Launching Other Projects From a Broadcast Step

You can broadcast a step that includes an inline project or which chains a project on the step's passage or failure (**Pass Chain** / **Fail Chain**). When you broadcast a step that launches (chains) another project, be aware that the broadcast step does not override the launched project's selector. In general, use a library project (one that has no selector of its own) when launching a project from a broadcast step, if your intent is to launch the project on every server matching the broadcast step's selector.

If you do not use a library project, each copy of the broadcast step runs on a different server, but the inlined or chained project obeys its own selector, which may not choose the same server as the copy of the broadcast step. You can end up with each broadcast step running on a different server, while all of the steps from an inlined project run on the same server, multiple times.

Note

If your intent is to use **Broadcast** to launch a library once on each server that matches a selector, be sure to also set the **Sticky** option on the library, so that all of its steps (that use the default project server) run on the same server.

Threading: Running Steps in Parallel

You can use threading to reduce job times, by running some steps from a job in parallel, on the same server or on multiple servers.

When a step's Thread property is set to Yes, the system attempts to run the step in parallel. Such a step is considered *thread-enabled* , and the step can be run as a separate thread (process) while the rest of the job continues. Threading obeys the following rules.

- At least two steps in sequence must have the Thread property set to Yes for threading to occur.
- When the system encounters a thread-enabled step, it attempts to start the step. If the following step is also threaded, the system attempts to start that step and continues on to the next step, until it hits the server's job limit, or it encounters a step whose Thread property is set to No.

  ### Note

  The actual start times of threaded steps depend on whether the step's resources are available; if a step cannot be started, the system waits and tries again. When you set up threaded steps, they run in parallel, and you cannot control which steps start first; the system decides that based on the circumstances of the servers at run time.

- The system chooses a server for each threaded step using the selector for the step. Steps may end up running simultaneously on one server (depending on the capacity of that server) or on several servers (depending on how many servers match the selector). However, you can force steps to run on a single server, by using the **Sticky** property. See “Making Steps Stick with a Server” on page 51
- Use the **Join** option for the step Thread property to separate threaded blocks of steps. The first set of threaded steps must complete before the next set of threaded steps can start.

Document # SC23-7681-07

In the following example, steps 2, 3, and 4 must complete before steps 5 and 6 can start. The Join option may be selected as the value for the Thread property during step creation.

Project	Thread Property for Step
Step 1	No
Step 2	Yes
Step 3	Yes
Step 4	Join
Step 5	Yes
Step 6	Yes
Step 7	No

- Use the **Max Threads** property for your project to keep the project from using too many system resources. Each thread-enabled step and any inline projects (which themselves might launch thread-enabled steps) can result in parallel processes, but all of those processes are counted against the maximum for the parent project. The system stops launching new parallel processes when it reaches the Max Threads value, and waits until the number of parallel processes for the project drops below the Max Threads value before continuing.

Log Filters

Use log filters to change the success criteria for a step. If filtering is not set up, Build Forge determines the success or failure of any step command by its exit status, where 0 is success and 1 is failure. Log filters allow you to evaluate step output and set step results to Fail, Pass, or Warn based on the contents of the step output, giving you the ability to closely control the criteria used to determine step success or failure.

For example, some commands always return a 0 exit status. A reporting command like `net use` prints a list of mapped network drives. The command always succeeds, even if the list does not contain the desired drive. Using a filter set, you can parse step output to look for a specific drive and mark the step as successful if it is found.

This section gives you information about log filters:

Creating a Log Filter

Log filters may contain one or more filter patterns. Each filter pattern is associated with an action, and optionally an access group for notification. You define a log filter first and then associate the log filter with a step in the project.

To create a log filter, do the following:

1. Select **Projects** → **Log Filters** .

   The Management Console displays the list of Log Filters and the New Log Filter form.

2. At **Name** , enter a name for the log filter, then click **Save** .

Document # SC23-7681-07

The Management Console saves the log filter and displays the New Pattern form.

3. For each filter pattern that you define for the log filter, do the following:

   a. Enter a regular expression in the **Pattern** field. (The regular expression must be Perl-compatible.) Build Forge searches step output for the pattern when the project runs. For details, see “Filter Patterns” on page 80.

   b. At **Action** , choose a filter action to take when the filter pattern is found. The default property, Set Fail, sets the step status to Fail. For details, see Filter Actions.

   c. At Notify, optionally select an access group to send members a notification e-mail when the filter is activated.

   d. Click **Save** .

To use the log filter, choose a project step and set the step's **Result** property to the new log filter. See Assigning a Log Filter to a Step.

Assigning a Log Filter to a Step

To use the log filter, you must assign the log filter to a project step using the step Result property. When you assign a log filter to a step, the filters patterns in the log filter are run on the step output whenever the project runs.

Note that when you assign a log filter to a step, the step result that is set by the log filter overrides all other criteria for determining the success or failure of the step. This includes the exit status for the step commands or any step properties. For example, if the step run time exceeds the time specified by the step Timeout property, the step stops. But, its status is not considered a failure unless its associated log filter action causes it to be set to Fail.

To assign a log filter to a step, do the following:

1. Select **Projects** or **Libraries** to access the step.
2. Select the project or library that contains the step.
3. Select the step to open the step Details form.
4. At **Results**, select the log filter that you want to run each time the step executes.

Filter Patterns

A filter pattern defines the character string or expression that you want to match in step output. Each filter pattern you create is associated with a single filter action. Both filter patterns and actions are defined in filter log sets. The ability to include multiple filter patterns in a log filter and apply it to output from a single step allows you to use multiple search criteria without constructing complex expressions.

To create a log filter, select **Projects** → **Log Filters** . For details, see Creating a Log Filter.

Filter Pattern Syntax

Review these guidelines for creating filter patterns:

- The filter pattern is defined as a regular expression and must use Perl-compatible syntax. For details about constructing Perl-compatible expressions, refer to Perl documentation.
- The system adds the delimiting forward slash characters (/<expression>/), so specify the expression **without** surrounding forward slash delimiters (expression).
- If your expression includes a metacharacter (for example, a /b), the metacharacter must be preceded by a backslash escape character (a\/b).

Syntax for some standard regular expressions are shown in the following table.

Expression	Matches
Production	Matches the *Production* anywhere in the string.
^Production	Matches *Production* at the beginning of the string.
Error:.*[0-9]$	Matches a line that contains *Error* followed by any set of characters terminated by a number at the end of the string.
[Ww]arning	Matches *Warning* or *warning*.
.*	Matches 0 or more of any character. The dot (.) matches any character, and the asterisk (*) matches any character 0 or more times.

Multiple Pattern Matches on the Same Line

To construct a pattern filter, it is important to understand how the system searches for pattern matches.

For each line of output, the system checks for matches against all the filter patterns in order; it stops when it finds a match, and moves on to the next pattern. So, if the pattern occurs twice on one line, the system may not find it.

```
exception retrying exception
```

For example, using the previous line of step output and the filter patterns in the following table, the system would match the first *exception*, set the step result to Fail, match *retrying* and set the step result to Pass, and move on to the next line without matching the second *exception*.

One way to resolve this problem is to replace the filter patterns in the table with the following filter pattern:

```
retrying.*exception
```

Filter Patterns	Filter Actions	Example Description
[Ee]xception [Rr]etrying	Set Fail - Fail Clear Fail - Pass	This is useful for Java projects; it fails the step on exceptions, but clears the failure on a retry. If the retry fails, a new exception will be generated, so that the final state of the command is valid.

Document # SC23-7681-07

Filter Actions

Filter actions define what action is taken when a filter pattern is found in step output. Each filter pattern you create is associated with a single filter action. Both filter actions and patterns are defined in log filters.

To create a log filter, select **Projects** → **Log Filters** . For details, see Creating a Log Filter.

Filter Action	Definition	Step Results
Set Fail (the default)	When the system finds the filter pattern, it sets the step results status to Fail and continues searching the current line for filter patterns in the set.	Fail
Set Fail/Halt	When the system finds the filter pattern, it sets the step results status to Fail, stops searching the current line for the filter patterns in the set, skips to the next line, and starts the pattern search again.	Fail
Clear Fail	When the system finds the filter pattern, it sets the step results status to Pass and continues searching the current line for the filter patterns in the set.	Pass
Clear Fail/Halt	When the system finds the filter pattern, it sets the step results status to Pass, stops searching the current line for the filter patterns in the set, skips to the next line, and starts the pattern search again.	Pass
Halt	When the system finds the filter pattern, it stops searching the current line for the filter patterns in the set, skips to the next line, and starts the pattern search again. It does not change the step results status.	not applicable
Include	Include allows you to reference one or more log filters in another log filter. You specify the log filter you want to include in the Pattern field and select Include in the Action field.	not applicable
Warning	When the system finds the filter pattern, it sets the step results status to Warn and continues searching the current line for the filter patterns in the set.	Warning
Clear Warning	When the system finds the filter pattern, it sets the step results status to Pass and continues searching the current line for filter patterns in the set.	Pass
Clear Warning/Halt	When the system finds the filter pattern, it sets the step results status to Pass, stops searching the current line for the filter patterns in the set, skips to the next line, and starts the pattern search again.	Pass
Notify Changers	To use Notify Changers, an adaptor that creates a relationship list must be included in the project and the adaptor step must run before the step that contains the Notify Changers log filter.	not applicable

Document # SC23-7681-07

Filter Action	Definition	Step Results
	The adaptor relationship list pairs users and objects (such as changed files). For details, see the Adaptor XML Reference. After the adaptor runs and creates the relationship list, if a log filter with the Notify Changers action matches its filter pattern in a step output line, the line is scanned again to try to match objects in the relationship list. If an object match is found, the users paired with the object are sent e-mail notification. For example in the following line of step output, the object match for the filter pattern `Error` is MyFile.c. So, the users paired with the MyFile.c object in the relationship list are sent e-mail notification of the error. `Error: Invalid token on line 55 of MyFile.c`	

Filter Notification

For every filter pattern in the log filter, you can optionally set notification to send e-mail to an access group to notify members that a pattern filter for a step has been activated.

Error Thresholds

You can use a special environment variable, _ERROR_THRESHOLD, to establish thresholds for individual steps and/or for a project. The system then counts the number of filter matches, and either fails the step or project when the threshold value is met, or notes the fact that the threshold was met in the job notes.

For more information, see “Special Case Variables” on page 40.

Error and Warning Counts

If filters are associated with steps to determine whether the steps succeed or fail, the system displays the number of errors and warnings caught by the filters. In the **Jobs → Completed**) tab, the numbers appear in in the **Results** column in parentheses after the job result. The format is **(<** *fail matches* **> / <** *warning matches* **>)**.

Example: A job result of **Failed (1 / 0)** shows that the job failed, 1 Fail filter was matched, and no Warning filters were matched.

Registers

Registers are general-purpose buffers that steps can use for storing persistent data. Ordinary registers can have single-letter names, or multi-character names that begin with letters.

All register names are stored in uppercase. Although you can create a register named Alpha, it is stored as ALPHA and must be referenced in project steps or in the .pop command as ALPHA.

Document # SC23-7681-07

You can include register variables in notification templates; use the ${X} braced form when referencing registers in notification templates. Referencing an empty register returns an empty string.

Use the .push and .pop dot commands to store information in and retrieve it from registers. See also the .poptag command (".poptag" on page 101), which makes the current job tag equal the contents of a register.

Note

Registers cannot be used in commands like variables. You must first pop the value of a register to a file before you can use it.

Special Registers

Register	Contains
!	Contains the command output lines that matched Fail filter patterns.
@	Contains the command output lines that matched Pass filter patterns.
=	Specifies the notes database for a job. Allows steps to add data from a file as a note to a job. This register is different from the others: • You can only write (push) to this register; you cannot read from it. • Data pushed to this register is always appended to it, rather than overwriting previous data. • The system supplies a time stamp and user ID with the appended data. This preserves an audit trail of job notes.

Project Registers

Project registers are distinct from ordinary registers. They persist across builds and can be created and viewed through the Management Console interface, making them an ideal way to store some kinds of configuration information.

For example, you could store an IBM® Rational® ClearCase® config spec as a project register, then have a step use a `.pop -p` command to extract the specification and use it with a cleartool `setcs` command, configuring your build. This allows you to manage the configuration along with the project.

If you have a project register named ALPHA, you could also have an ordinary register named ALPHA, with entirely different contents. Project registers are a separate set of values.

You can create and access project registers in two ways:

- Through dot commands (`.push` and `.pop`), by adding a -p option. When you use the -p option, your command refers to a project register, rather than an ordinary register.

  For example, a command

Document # SC23-7681-07

```
.push -p ALPHA register.txt
```

places the contents of the file register.txt in the project register named ALPHA.

- Through the Management Console interface. Select **Projects** , then click the icon next to the desired project's name. The project properties appear in the lower pane; click the **Registers** tab to display the project's registers. The tab provides a form for managing registers:
  - To create a new register, enter a name and contents, then click **Create** .
  - To delete a register, click the trash can icon next to the register's name in the list at the right of the form.
  - To edit a register, click the icon next to the register's name in the list. The system populates the register form with the register's contents. Make your changes, then click the **Save Edited Register** button.

Anyone who has access to a project can view and edit its project registers.

Note

Use uppercase to create register names and to reference all register types. Even though you can create registers using lowercase letters, registers are stored and must be referenced using uppercase letters.

Dot Commands

The system includes a set of special commands called dot commands. The system intercepts commands that are preceded by a period and uses them to perform special functions in the system.

Most dot commands must be used in the command field of a step, but a few, the environment dot commands, can be used in environment variables. See “Environment Dot Commands” on page 113 for instructions.

You can mix dot commands with ordinary commands in a step and you can have multiple dot commands in a single step. Do not use more than one .scan command in a single step, however, as the system cannot accurately report the command's results if you do.

Dot Command Reference

Dot commands are commands you can embed in the Command field of a step, which are intercepted by the system and reprocessed or acted on rather than being sent directly to the command interpreter of the server. They provide access to special capabilities and functions within the system. In some cases the dot commands provide platform-independent methods of performing actions; in others, they are used to manipulate data.

Each dot command includes one or more syntax lines to describe the command format. The syntax lines use the following notation:

- User-supplied values: <Shown in angle brackets>

Document # SC23-7681-07

- Optional text: [Shown in brackets]

You can use environment variables for command parameters except where specifically noted.

.agentupdate

Description

Use the .agentupdate command to upgrade an agent to a later version. The .agentupdate command copies the latest version of the agent executable file from the Management Console to the Build Forge server.

Syntax

```
.agentupdate <directory_path> agent_executable_file
```

The name of the *agent_executable_file* for the Build Forge agent is required. This file is sent from the Management Console to the Build Forge server.

- For Windows, specify BFagent.exe.
- For Unix, specify bfagent.

If you specify the file name only, the .agentupdate command looks for the executable file in the default installation directory on the Management Console host:

- (Windows default) C:\Program Files\IBM\Build Forge
- (UNIX/Linux default) usr/local/buildforge

Use the <*directory_path*> if you specify a non-default installation path. Build Forge looks for the executable file in this directory.

Examples

```
.agentupdate BFagent.exe

.agentupdate bfagent

.agentupdate /usr/lib/bfagent
```

Prerequisites

For Windows agents, before running .agentupdate, do or note the following:

- Verify that the user registered in Server Auth has write permission on <*bf-install*>\Agent.
- On Windows, the BFagent.exe is copied directly to <*bf-install*>\Agent; an update directory is not used.

For Unix agents, before running .agentupdate, do or verify the following:

- Specify an update directory in the /etc/bfagent.conf file (the default is the etc directory). In the bfagent.conf file, locate the update_path line, then specify the full path name, including the executable file name (bfagent). The

Document # SC23-7681-07

.agentupdate command copies the bfagent executable from the Management Console to the update directory.

- Verify that the user registered in Server Auth has write permission on the update directory.

Upgrading Multiple Agents

To use the .agentupdate command to update the same Build Forge agent on every Build Forge server on which the agent is installed, use the step Broadcast property as follows:

- Create a selector for the .agentupdate step. The selector defines the properties for the Build Forge servers whose agents are to be updated.
- Set the Broadcast property for the .agentupdate step to Yes.

.bom

```
.bom addcategory "<Category Name>"
.bom setcolumn "<Category Name>" "<SectId>" "<Column>" "<Column>" "<Column>"
.bom data <Category Name>" "<SectId>" "<Column=Value>" "<Column=Value>" "<Column=Value>"
```

The .bom command adds data to the Bill of Materials (BOM) for a build. With it, you can add categories, sections, and rows and columns of data.

Categories — A category is a visible and expandable header within the BOM. It contains a table of data.

Sections — A section is a logical grouping of rows within a category. Section names are not displayed, but the system prints a new set of column headers to mark each section.

The command has three options:

- Use the `addcategory` option to create a new category in the BOM. For example:

  ```
  .bom addcategory "Spaceships"
  ```

  creates a category named Spaceships.

- Use the `setcolumn` option to specify sections and columns of data in a category. You specify a section ID and then column headers. For example:

  ```
  .bom setcolumn "Spaceships" "Section1" "ShipName" "WarpSpeed" "Tonnage"
  ```

  sets up a table of data about spaceships.

- Use the `data` option to specify rows of data. For example:

  ```
  .bom data "Spaceships" "Section1" "ShipName=SpaceShipOne" "WarpSpeed=9" "Tonnage=10000"
  ```

  specifies that SpaceShipOne's warp speed is 9 and its tonnage is 10,000.

Document # SC23-7681-07

As with other dot commands, you can use environment variables in the command. A command like

```
.bom data "Spaceships" "${SECTION}" "ShipName=${NAME}" "WarpSpeed=${SPEED}"
"Tonnage=${TONNAGE}"
```

populates the BOM with data loaded into environment variables by earlier commands.

You can create any number of columns, but the system does not write a line to the BOM until the last column is populated. Also, if you omit a column from a data line, the system uses the value from the previous row. Using the Spaceships example, if you added a line

```
.bom data "Spaceships" "Section1" "ShipName=Freighter" "Tonnage=20000"
```

the system would repeat the WarpSpeed value, giving rows like these:

```
ShipName      WarpSpeed   Tonnage
SpaceShipOne      9        10000
Freighter         9       20000
```

.bomexport

Description

The .bomexport dot command exports the BOM for the job to an XML file. After collecting the BOM information, .bomexport saves it to the file and location you specify.

The path and file name are optional. By default, Build Forge saves the BOM report to the step's working directory on the server and uses the tag name as the file name (<*build_tagname*>.xml).

Specify the .bomexport command as the last step in the project.

Syntax

```
.bomexport [path_name][file_name]
```

Options

Option	Description
path_name	An optional path name. If provided, the path must be relative to the step's working directory on the Build Forge server. If omitted, the file is saved to the step's working directory.
file_name	An optional file name. The BOM for the job is saved to the file in XML format. If a file name is not provided, a file name is constructed from the build tag name (<*build_tagname*>.xml).

Examples

```
.bomexport
```

```
.bomexport myproj.xml
```

```
.bomexport path/to/myproj.xml
```


Document # SC23-7681-07

```
.bomexport /path/to/myproj.xml
```

.bset

```
.bset env "<VariableName>=<DesiredValue>" ["<VariableName>=<DesiredValue>"]

.bset selector <SelectorName>
.bset server <ServerName>
.bset buildserver <ServerName>
```

The .bset command changes project settings temporarily during a job. The command has several options:

- Use the *env* option to change the value of one or more project environment variables during a job. For more information on using the command this way, see “Changing Environment Variable Values During a Job” on page 111. You can use .bset to set a variable that does not yet exist. The values set by the .bset command are written to the job record; they do not update the database record for the environment set. Later jobs are not affected by the changes, since they get new copies of the environment set from the database when they launch.
- Use the *selector* option to change the project selector during a job. When you change the selector, any steps that follow the step that contains the .bset selector command use the new project selector, if they do not have a selector setting of their own (their selector field is set to default).
- Use the *server* option to change the containing steps project server during a job. This option has no effect unless the project's **Sticky** property is set. Steps after the .bset server command will use the new default server (if they do not have a selector setting of their own). This allows you to have a project stick to one server for part of the project, then stick to a different server for the remainder.
- Use the *buildserver* option to change the default job server during a job for project steps that follow the .bset buildserver command. The buildserver option applies to inline and project-level steps that are set to use the default selector.

The **.bset selector -p** option changes the selector for steps in the currently executing project, and any subsequent inlines that the steps call. It is commonly used in inlined projects when the intent is to change the selector for this project and its steps and descendants (other inlines), but not the project that called the currently executing project.

- **.bset selector** will change the default selector for the entire job.
- **.bset selector -p** will change the selector for just that part of the job that contains the steps project, for the sibling steps and child inlined projects of the sibling steps.

For example:

```
JOB
  Step A
        Inline AA
        Step B
```


Document # SC23-7681-07

```
              Inline BB
              Step C
  Step D
        Inline DD
        Step E
```

If Step B is **.bset selector** , it will change the default selector for **JOB** . If Step B is **.bset selector -p** , it will just change the default selector for Inline A and its children (Inline BB and Step C). Step D/E will still use the original default selector.

The .bset changes do not take effect until the current step completes. For example, if your step includes multiple commands as in the following example, the second line operates in an environment unmodified by the .bset command:

```
.bset env "CompilerVersion=1.1"
compile driverset
```

When the second line executes, the CompilerVersion value has not been set. To remedy this, move the compile command to the following step.

You can use backtick syntax to set the server name to the output of a command. For example, the command

```
.bset server `SelectAServer.sh`
```

runs the SelectAServer.sh script and provides its output as the server name for the .bset server command.

Note

Using multiple .bset commands in threaded steps is not recommended, in order to avoid race conditions when attempting to set the same variable, server, or selector. Instead, use **.bset selector -p** .

.break

```
.break [<notification_group_name>]
```

Use the .break command to make a job halt until you restart it. When the system encounters a step with a .break command, the run completes with a result of Stop. Use the **Restart** icon to restart the job, continuing with the step after the .break step.

If the .break command occurs within a chained job, the system stops the chained job, but returns control to the calling job, which continues processing steps.

You can include an access group as an optional argument to the command; if you do, the system sends an e-mail message to the specified access group when it stops the job.

.date

```
.date <date_format_code>
```


Document # SC23-7681-07

The .date command is used within an environment variable, as the contents of the variable. When it is used, the system replaces the variable value with the current date and/or time value at run time. Date format codes supplied to the .date command determine what date/time information is actually supplied.

To use the .date command, define an environment variable with a value of ".date <*date_format_code*>." It cannot be used in step command fields.

Note

The .date command is processed by the Management Console and thus uses the time and time zone values on the Management Console machine.

For example, if you define a variable named DayOfWeek with a value ".date %A", assign the environment containing that variable to your project, and run the project on a Wednesday, then the system assigns the text "Wednesday" to the variable DayOfWeek.

The following list shows commonly-valid format strings for the .date command. The format strings are built on the POSIX strftime() function. Specific support for formatting parameters depends on the underlying operating system.

Format String	Description
%a	The abbreviated weekday name according to the current locale.
%A	The full weekday name according to the current locale.
%b	The abbreviated month name according to the current locale.
%B	The full month name according to the current locale.
%c	The preferred date and time representation for the current locale.
%d	The day of the month as a decimal number (range 01 to 31).
%H	The hour as a decimal number using a 24-hour clock (range 00 to 23).
%I	The hour as a decimal number using a 12-hour clock (range 01 to 12).
%j	The day of the year as a decimal number (range 001 to 366).
%m	The month as a decimal number (range 01 to 12).
%M	The minute as a decimal number (range 00 to 59).
%p	Either "AM" or "PM" according to the given time value, or the corresponding strings for the current locale. Noon is treated as "pm" and midnight as "am".
%S	The second as a decimal number (range 00 to 61).
%u	The day of the week as a decimal, range 1 to 7, Monday being 1.
%U	The week number of the current year as a decimal number, range 00 to 53, starting with the first Sunday as the first day of week 01.
%w	The day of the week as a decimal, range 0 to 6, Sunday being 0.
%W	The week number of the current year as a decimal number, range 00 to 53, starting with the first Monday as the first day of week 01.
%y	The year as a decimal number without a century (range 00 to 99).
%Y	The year as a decimal number including the century.

Document # SC23-7681-07

Format String	Description
%Z	The time zone or name or abbreviation.
%%	A literal “%” character.

.defect

Description Use the .defect command to add an adaptor for a defect tracking application to a project step. A defect adaptor is a Build Forge object; it is based on the adaptor template for a defect tracking application. The adaptor code for the step is executed when the project runs.

Syntax `.defect <adaptor_name> [entry_name]`

The *<adaptor_name>* is required; it is the name assigned to the adaptor in the Management Console. The *<adaptor_name>* case should match the case used in the console.
If your adaptor template has multiple interface functions, specify the one that you want to execute by using the *entry_name* option. The *entry_name* must match the name attribute specified for interface element in your adaptor template. In the following example, the entry name is DefectFunction.

Examples
```
.defect MyClearCaseQuestAdaptor
```

```
.defect MyClearCaseQuestAdaptor DefectFunction
```

Notes To create an adaptor or view a list of adaptors, select **Projects** → **Adaptors** .
The adaptor templates provided with the Build Forge product are located in:

`<bf-install>/interface`

.drill

```
.drill [through] <"var1,var2,var3"|$EnvVar|-r[p] Register> [gr[ouped by] "{}"] [sep[arated
 by] ","] [exec] "Command $1 $2"
```

The .drill command allows you to loop over a command, executing the command once for each member of a series of values. You can specify the values on the command line, or draw them from an environment variable or register. When it executes a .drill command, the system uses the .drill syntax to construct a series of command lines and sends them to the agent for execution.

For example, the command `.drill "A,B,C,D" "echo value $1"` creates the following commands:

```
echo value A
echo value B
echo value C
echo value D
```

Grouping

You can group the values and reference multiple values in each group using the $1, $2, etc. syntax. $1 refers to the first value in the group, $2 to the second value in the group. For example, `.drill`

Document # SC23-7681-07

`through "(A,B,C,D,E),(B,C,D,E,F),(C,D,E,F,G)" grouped by "()" separated by "," exec "echo 1[$1] 2[$2] 3[$3] 4[$4] 5[$5]"` creates these commands:

```
echo 1[A] 2[B] 3[C] 4[D] 5[E]
echo 1[B] 2[C] 3[D] 4[E] 5[F]
echo 1[C] 2[D] 3[E] 4[F] 5[G]
```

Note

There is no default grouping character. There is a default separator character, the comma. If you do not specify `grouped by`, the system looks through the supplied values as separated by the separator character and considers each such string a single value. For example, the command `.drill "(A,B),(C,D)" "echo $1 $2"` resolves to the following commands:

```
echo (A 2
echo B) 2
echo (C 2
echo D) 2
```

Data Sources

You have several options for where the .drill command gets the data that it loops through. The first parameter for the command is the data source. You can include the optional command word "through" to indicate the data source.

- You can explicitly list the data in the command line, as in the following command, which loops over the values one, two, and three:

  ```
  .drill through "one,two,three" exec "echo $1"
  ```

- You can draw the data from an environment variable. The following command assumes that the environment variable FILENAMES is a comma-separated list of files, and uses a DOS command to delete all the files in the list:

  ```
  .drill through $FILENAMES exec "del $1"
  ```

- You can draw the data from a register or a project register. If RegisterA contains a comma-separated list of filenames, then the following command issued to a Linux system writes out the content of each file:

  ```
  .drill -r RegisterA exec "cat $1"
  ```

  while the following example does the same but uses a project register:

  ```
  .drill -rp ProjectRegisterA exec "cat $1"
  ```

.edit

`.edit /<search_expression>/<replace_expression/ [<relative_path>/]file [file ...]`

Use the .edit command to search and replace text strings in one or more files. The .edit command replaces the first instance of the string (search_expression) on each and every line in each file

Document # SC23-7681-07

specified. Files are assumed to reside in the step's working directory unless you specify a relative path.

The .edit command implements standard POSIX regular expressions for matching and replacement, including the use of () substring selection and \N substitution in the replacement pattern.

.edit replaces the first instance of the string on each and every line in each of the files.

Note

You must explicitly list one or more file names, without wildcards.

For example, the following command replaces strings such as winXPdriver and win2000driver in the file called drivermakefile.

```
.edit /win*driver/linuxdriver/drivermakefile
```

The .edit command is similar to the .strsub command; differences include:

- The .strsub command is faster than .edit when performing replacements in large text files or many files.
- The .edit command can perform regular expression searches and replaces.
- The .edit command replaces the first instance only of the string (search_expression) on each and every line in each file.
- The .strsub command replaces every instance of the string (source) on each and every line in each file.

.export

```
.export [path_name][file_name]
```

The .export command saves a copy of the current project to the current working directory in XML format. The XML file describes the project, current build, and steps. It does not describe other associated objects, such as the server.

The exported XML file can be used to import the project definition to the Management console.

The .export command can take an optional path and/or file name. The path must be a relative path. It is applied from the step's working directory.

If no file name is provided, the file name is constructed from the current project tag: $BF_TAG.xml.

.get

```
.get server:[[<relative_path>/]file/]file [[[<relative_path>/]file/]file]
```

Use the .get command to transfer a file from one logical server to another. The .get operation executes from the current server/path and retrieves the file from the specified server/path. The destination path name is relative to the current step's working directory. The source path name is relative to the specified server's base path. Server must specify a logical server that allows the

Document # SC23-7681-07

.get operation for files (see “Enabling File Copying on a Server” on page 28). Only single files may be transferred.

You can include environment variables in a .get command, so that the paths you use can be relative to the environment used by a specific job.

If the server name you are using has spaces in it, wrap the name in quotes.

The transfer is not fast, so you may want to choose a different method to transfer large files. Expect speeds of no more than 40 KB per second; a 70 MB file could take 45 minutes to an hour to transfer.

Note

If the destination file already exists, it is overwritten without warning. Also see the description of paths in “Working Directories for Jobs” on page 129.

.include

The .include command is an environment dot command. You use it by creating an environment variable named .include within an environment, and setting the value of the variable to the name of another environment. The system then includes all of the variables from the referenced environment in the environment that contains the .include.

Note

You can use more than one .include variable in an environment, to include as many other environments as desired. You can use this feature to create nested sets of variables, or to manage a variable in a single location yet use it within many other environments.

.load

```
.load [-o] [-e] [-v] [-j] [<relative_path>/]<filename>
.load -r|-p <registername>
.load -s `<command name>`
```

The .load command loads a project from an XML file and adds the steps of the loaded project to the current project, *after* the step that executed the .load command, allowing a project to dynamically create and load steps at run time. Using options, you can cause the .load command to draw its data from a register or from the output of a command.

To write an XML file for a .load command, start with an export file from an existing project to give you the appropriate basic structure. You can also create a project within the system, then export it to use it in a .load command. This topic includes sample XML code.

The steps loaded by a .load command can contain references to inlined or chained projects. By default, the system looks for the definitions of inlined projects within the XML file, and loads their steps; see the -e option below for a way to have the system get the inlined project definition from the database. For pass chained or fail chained projects, the system always looks for the project definition in the database.

Document # SC23-7681-07

Multiple Projects in XML Files

Because the system exports inlined projects along with their calling projects, an XML file may contain several projects. The .load command executes the project that is labelled primary in the file. This project has attribute `primary="1"` on its `<project>` element.

Command Options and Parameters

The simplest command form is `.load <filename>`. You can include an optional path name (relative to the job directory) in front of the filename. For example, the command

```
.load ../../project.xml
```

loads the file project.xml from the server directory (the directory that contains the project and job directories), assuming that the step's path property is "/" (the default).

Note

When a normal step launches an inlined project, the system goes to the database to the get the current definition for that project; when a step that is imported by .load command launches an inlined project, the system looks inside the XML file for the definition of the inlined project. See the description of the -e option below for a way to avoid this situation.

The command has the following options:

-r or -p — These options cause the system to load steps from a register. Use the command line with these options.

```
.load -r|-p <registername>
```

The -r option loads steps from an ordinary register, while the -p option loads steps from a project register. You can build up data in a register in earlier steps in your project, then load the steps from the register with this command.

-s — This option causes the system to run a command and use the output of that command as the data to load. Use the command line

```
.load -s `<command name>`
```

-e — When the -e option is set, the system gets inlined projects from the database instead of from the loaded XML file. It treats the value of `chainID` as a reference to a project ID within the database. This enables your XML file to reference the latest version of an inlined project, instead of the one in the XML file, or to reference a project that is not included in the XML file.

-o — Use the -o option to disable inlined projects within the XML file. When this option is used, the system ignores any inlined projects within the main project. A step that contains a reference to an inlined project executes its command but then ignores its inline.

Document # SC23-7681-07

-j Use the -j option if the last set of steps in the XML file are threaded and the steps following the .load command are also threaded. The -j option turns the last threaded step into a join step. Otherwise, the threaded steps become part of the threaded block of steps following the .load command.

Document # SC23-7681-07

Example 2. Sample XML

The following example shows an XML file to use with the .load command. The XML was created by exporting a project named HelloWorldPlusInline.

Note the following details of the example XML:

- The XML contains two `<project>` elements.
- The first project in the XML is the primary project; it has the attributes `name="HelloWorldPlusInline"` and `primary="1"` .
- The second project in the XML is called Sleepytime and has the attribute `primary="0"` to indicate that it is not primary.
- The first step of the HelloWorldPlusInline is a step named EchoHelloWorld which contains an echo command and a `chainID` attribute. The `chainID` attribute has a value of 2, indicating that the system should inline the project with the ID 2, which is the Sleepytime project.

  ### Note

  Ignore the step attribute `inline`; it is a deprecated attribute that is no longer used. All steps have this attribute with a value of N. To determine if a step has an inlined project, look for the attribute `chainID` . The value of `chainID` refers to the ID of a project. By default, the system looks for the inlined project within the XML file, but if you use the -e option in your .load command, the system treats the value as project ID within the database. This allows you to create your own .load files without having to include inlined projects within them.

- Each project has an `id` attribute. This ID value is the same as the project's ID in the database. You can get a list of project IDs by executing the following command from your installation directory.

  ```
  bfexport -l
  ```

```
<?xml version="1.0" encoding="UTF-8"?>

<buildforge schema="7.000301" comment="">
  <project access="6" active="Y" name="HelloWorldPlusInline" primary="1"
selectorId="Choose_local" maxthread="0" increment="Y" tagsync="0" buildclass="Production"
 sticky="N" envId="0" tag="BUILD_$B" id="19" exclusive="0">
    <tagvar autoincrement="Y" name="B" id="1">2</tagvar>
    <step absolute="N" failwait="N" selectorId="" dir="/" broadcast="N" timeout="300"
id="1" passwait="N" inline="N" threadable="N" chainId="2" access="6" active="Y"
passnotify="0" description="EchoHelloWorld" onfail=" " failnotify="0" envId="0">
      <command>echo Hello World</command>
    </step>
    <step absolute="N" failwait="N" selectorId="" dir="/" broadcast="N" timeout="300"
id="2" passwait="N" inline="N" threadable="N" access="6" active="Y" passnotify="0"
description="export proj to build and server folders" onfail=" " failnotify="0" envId="0">

      <command>.export $BF_PROJECTNAME_PHYS.xml
copy /Y $BF_PROJECTNAME_PHYS.xml ..\..</command>
    </step>
```


Document # SC23-7681-07

```
  </project>
  <project access="6" active="Y" name="Sleepytime" primary="0" selectorId="Choose_local"
maxthread="0" increment="Y" tagsync="0" buildclass="Production" sticky="N" envId="0"
tag="SLEEP_$B" id="2" exclusive="0">
    <tagvar autoincrement="Y" name="B" id="1">21</tagvar>
    <step absolute="N" failwait="N" selectorId="" dir="/" broadcast="N" timeout="300"
id="1" passwait="N" inline="N" threadable="N" access="6" active="Y" passnotify="0"
description="Sleep, perchance to dream" onfail=" " failnotify="0" envId="0">
      <command>.sleep 0</command>
    </step>
  </project>
  <class maxdays="0" access="1" entranceprojectId="1" name="Production" keepfiles="B"
deletechangedata="N" purgeprojectId="2" exitProjectId="5" candidates="AnyBuild "
maxbuilds="0"></class>
  <selector operator="" required="" access="6" value="" name="Choose_local" selectorId=""
 property=""></selector>
</buildforge>
```

.lock

```
.lock
```

The .lock command causes the system to lock a job after it completes. This prevents the job from being automatically deleted based on the properties of its class; also, a locked run is not listed on the **Jobs → Completed** tab, appearing instead on the **Locked** tab. The command takes no parameters; it locks the job that it is used in.

.mkdir

```
.mkdir <relative_path>
```

The .mkdir command creates a directory. The `<relatative_path>` parameter is interpreted as a relative path from the current step directory. If directories in the path name specification do not exist, they are created. Absolute paths and paths including a drive letter (like `c:\`) are not allowed.

.monitor

```
.monitor [-c] [-w] <interval> [<relative_path>/]<filename>
```

The .monitor command causes the system to halt the project while it watches a file to see when the file size stops changing. When a step issues this command, the system checks the indicated file; it then rechecks the file every <interval> seconds. When the file size fails to change between two intervals, the system continues to the next step.

If you use the -c option, the system writes the contents of the monitored file out to the step log after it determines that it has stopped changing; then it continues on to the next step.

If the file does not exist, then the system does not wait but continues immediately after the first interval. Use the -w option to force the system to wait for the file to be created before starting the monitoring process.

Document # SC23-7681-07

.pack

Description Use the .pack command to add an adaptor for a packaging application to a project step. A packaging adaptor is a Build Forge object; it is based on the adaptor template for a packaging application. The adaptor code for the step is executed when the project runs.

Syntax `.pack <adaptor_name> [entry_name]`

The *<adaptor_name>* is required; it is the name assigned to the adaptor in the Management Console. The *<adaptor_name>* case should match the case used in the console.
If your adaptor template has multiple interface functions, specify the one that you want to execute by using the *entry_name* option. The *entry_name* must match the name attribute specified for interface element in your adaptor template. In the following example, the entry name is PackageFunction.
To execute a different interface, edit the adaptor template to set the default attribute to true (default="true") on the interface that you want to execute.

Examples
```
.pack MyPackagingAdaptor
```

```
.pack MyPackagingAdaptor PackageFunction
```

Notes To create an adaptor or view a list of adaptors, select **Projects → Adaptors** .
The adaptor templates provided with the Build Forge product are located in:

`<bf-install>/interface`

.pop

```
.pop [-p] <register_name> [+][<relative_pathname>|-]
.pop [-p] <register_name> [>|>>]<register_name>
```

Write the contents of a register to a file, to the step log, or to another register.

The optional -p parameter makes the command refer to a project register. Project registers are separate from ordinary registers, and project registers persist after a job ends.

Use uppercase to create register names and to reference all register types. Even though you can create registers using lowercase letters, registers are stored and must be referenced using uppercase letters.

The following examples show the variety of uses for a .pop command:

- `.pop A data.txt` - writes register A to the file data.txt, located in the step's working directory.
- `.pop VER +data.txt` - appends the contents of register VER to the file data.txt.
- `.pop ALPHA` - writes the contents of register ALPHA to the step's log.
- `.pop ALPHA > BETA` - gives register BETA the same contents as register ALPHA.

Document # SC23-7681-07

- `.pop A >> B` - appends the contents of register A to register B.

Note

Popping a register does not empty it. To change the contents of the register, push a new value into it using the `.push` command.

.poptag

```
.poptag [-p]<registername>
```

The .poptag command changes the current tag, replacing it with the contents of the specified register.

The optional -p parameter makes the command refer to a project register. Project registers are separate from ordinary registers, and project registers persist after a job ends.

Use uppercase to create register names and to reference all register types. Even though you can create registers using lowercase letters, registers are stored and must be referenced using uppercase letters.

.purge

```
.purge
```

The .purge command sets a flag that causes the job to be purged immediately after the job completes. A .lock command that executes after the .purge command causes the job to be saved instead. You can use this command to create jobs that are saved only if they successfully complete all of their steps, by making a .purge command the first step in the project, and a .lock command the last step.

.push

```
.push [-p] <register_name> <relative_pathname>
```

Push the contents of <relative_pathname> into register <register>. The register name may be prefixed with a “+” indicating that the contents of the file are to be appended to the register, and you can replace the <relative_pathname> with a hyphen to clear the register.

The optional -p parameter makes the command refer to a project register. Project registers are separate from ordinary registers, and project registers persist after a job ends.

Use uppercase to create register names and to reference all register types. Even though you can create registers using lowercase letters, registers are stored and must be referenced using uppercase letters.

The <relative_pathname> uses the project/tag path unless the Absolute property for the step is enabled.

The following examples assume that the Absolute property is not enabled for the step:

Document # SC23-7681-07

- `.push ALPHA data.txt` - places the contents of the file data.txt (in the step's working directory) in register ALPHA.
- `.push +B ..\newdata.txt` - appends the contents of the file newdata.txt (in the parent directory of the step's working directory) to register B.
- `.push ALPHA` - clears register ALPHA.

.put

`.put [<relative_path>/]file server:[[<relative_path>/]file]`

Use the .put command to transfer a file from one logical server to another. The .put operation executes from the current server/path and sends the specified file to the remote server. The destination path name is relative to the target server's base path. The source path name is relative to the step's current working directory. The remote server must specify a logical server that allows the .put operation for files (see “Enabling File Copying on a Server” on page 28). Only single files may be transferred.

You can include environment variables in a .put command, so that the paths you use can be relative to the environment used in a specific job.

If the server name you are using has spaces in it, wrap the name in quotes.

Note

If the destination file already exists, it is overwritten without warning. Also see the description of paths in “Working Directories for Jobs” on page 129.

.retag

`.retag <new_tag>`

Use the .retag command in a step to change the tag for a job during the job. You can use variables or commands as the new tag value.

.retry

`.retry <count> <command>`

The .retry command allows a command to be retried on failure. The .retry command takes a single count argument that specifies the number of times to retry the command. The command to execute is taken as the remainder of the arguments and thus the .retry command must be the final dot command for a step. For example, the command

```
.retry 3 myscript.sh arg1 arg2 arg3
```

executes “myscript.sh arg1 arg2 arg3” up to 3 times before failing the step. The first invocation of the command that returns a successful status stops the retry process.

Document # SC23-7681-07

.rget

```
.rget server:[<path>] [<path>]
```

The .rget command works like the .get command, but copies an entire directory tree, recursively. You must supply directory names as the parameters. For example, the command

```
.rget winbuildserver1:config myconfig
```

copies the contents of the directory config on the server winbuildserver into the myconfig directory on the current server.

Note

You cannot use environment variables in this command.

.rmdir

```
.rmdir <relative_path>
```

The .rmdir command removes a directory specified by <relative_path>. The system removes the base directory specified by the path name, including all contents and descendants.

.rput

```
.rput [<relative_path>] server:[<relative_path>]
```

The .rput command works like the .put command, but copies an entire directory tree, recursively. The relative paths you supply must be directories, not files. For example, the command

```
.rput myconfig linuxserver5:feb2005
```

copies the contents of a directory myconfig from the current server to the feb2005 directory on server linuxserver5.

Note

The source path is relative to the working directory of the step, so it includes or does not include the project and tag directories based on the value of the step's Absolute property. The destination path is relative only to the Path property of the destination server. See “Working Directories for Jobs” on page 129 for more information on how the system constructs paths.

Note

You cannot use environment variables in this command.

.run and .runwait

```
.run [-c "<condition>"] "<ProjectName>"
```

```
.runwait [-c "<condition>"] "<ProjectName>"
```


Document # SC23-7681-07

The .run and .runwait commands allow you to launch a chained project from a step command, providing a more flexible means of launching chains, including the evaluation of environment variables to determine whether a chain is launched.

The commands differ in how they behave after they launch a project:

- The .run command launches the specified project as a chain, following the rules for environment variable inheritance for chained projects.
- The .runwait command launches the specified project; then the launching step waits while the launched build completes. When the launched build completes, the system sets the result value of the launching step to the pass or fail status of the launched build (the only values that can result are pass or fail).

Important

A project that includes a step with .runwait consumes two job slots when it runs. If there are insufficient job slots available, the step fails with an error.

Conditional Launches

The optional -c parameter can be used to make the launch depend on a condition. You can use environment variables in the condition. The condition can be of several forms:

String comparison	You can use equal (=) or not equal (!=) operators to evaluate strings. The chain is launched if the comparison evaluates true.
Numeric comparison	You can use <, >, <>, ><, or = operators to compare two numeric values.
Command success	You can use a command enclosed in backticks as the value of the -c parameter. The system runs the command; if it succeeds, the chain is launched.

Example 3. Examples

```
.run "BuildWindowsDriver"
```

The system launches the BuildWindowsDriver project. The launching project continues with the next step immediately.

```
.runwait "BuildWindowsDriver"
```

The system launches the BuildWindowsDriver project. The system pauses the launching project at the .runwait step. When the BuildWindowsDriver project completes and passes, the .runwait step's status is set to pass.

```
.run -c "$HOMEDRIVE=C:" "Simple Echo"
```

The system runs the project Simple Echo if and only if the HOMEDRIVE variable has the value C:.

This command produces log output like the following (in the EXEC section of the step log):

Document # SC23-7681-07

- When HOMEDRIVE is C:

```
.run Condition: 'C:' = 'C:' satisfied.

Queueing Project "Simple Echo" on server [WinBox].
Queued Build 'BUILD_202' of project 'Simple Echo'.
```

- When HOMEDRIVE is not C:

```
.run -c "$HOMEDRIVE=C:" "Simple Echo"

.run Condition: 'D:' = 'C:' unsatisfied, no project queued.
```

The system can numerically compare strings if they contain numbers. For example, it handles the following cases as shown:

.runwait -c "a12b<c42d" "Simple Echo"

```
.run Condition: '12' < '42' satisfied.
Queueing Project "Simple Echo" on server [WinBox].
Waiting for .run build (4411) to complete.
.run build is now running.
.run build has finished.
Build 'BUILD_203' of project 'Simple Echo' completed.
```

.runwait -c "f43g<>h43i" "Simple Echo"

```
.run Condition: '43' <> '43' unsatisfied, no project
queued.
```

The following examples show how to use commands as conditions. Note that the command must be enclosed in both quotes and backticks:

.run -c "`exit 1`" "Simple Echo"

```
Env .run encountered an error during variable expansion,
parameter [`exit1`] expanded to [].
Expansion returned non-zero exit, project will not be
queued.
```

.run -c "`exit 0`" "Simple Echo"

```
Expansion returned zero exit, project will be queued.
Queueing Project "Simple Echo" on server [WinBox].
Queued Build 'BUILD_204' of project 'Simple Echo'.
```

When you use .runwait and a build fails, the log looks like the following:

.runwait "Fail Build"

```
Queueing Project "Fail Build" on server [WinBox].
Waiting for .run build (4413) to complete.
.run build is now running.
.run build has finished.
Build 'BUILD_3' of project 'Fail Build' Failed, setting step
 status to fail.
```

.scan

`.scan [-v][-i <ignorepattern>] baseline | checkpoint`

Document # SC23-7681-07

The .scan command enhances the data stored in the BOM for the job. It tracks the files in the step's working directory, along with MD5 values for each file. Use the command in one of two forms:

```
.scan baseline
```

Stores a list of all files in the step's working directory. The system displays the list as a category in the BOM for the job. You can have multiple baseline commands in a job, but each one resets the list to the state of the step's working directory when the .baseline command executes. The final BOM displays only one baseline category.

```
.scan checkpoint
```

Stores a list of all new, changed, and deleted files since the last .scan baseline or .scan checkpoint command in the job. As with the .scan baseline command, the system displays the list in the BOM, but each checkpoint command creates a new category in the BOM. You must issue a .scan baseline command before the first .scan checkpoint command in your job. A .scan checkpoint command that precedes a .scan baseline command is ignored.

The command has two options:

- The -v, or verbose, option causes the system to record a copy of the change information in the job log.
- The -i option allows the command to ignore directories that match the supplied pattern (a match at the beginning, end, or any directory portion of the path). Use this option to eliminate source control directories from the change listings. For example, you can use `.scan -i CVS checkpoint` to keep CVS directories out of the reports, or .scan -i .svn baseline to ignore Subversion directories. Note that the system still logs changes to these directories if you use the -v option, but they are not included in the BOM.

Note

Do not use more than one .scan command in a single step. The system cannot provide accurate output for the .scan commands if you use more than one in a single step.

For more information on using these commands, see “Using the Bill of Materials” on page 132.

.semget

```
.semget <semaphorename>
```

When a step issues this command, the system checks whether a semaphore with the stated name exists.

- If no such semaphore exists, the system creates one and assigns it to the step's job. Then execution continues with the next step.
- If some other job has already claimed this semaphore name, the job hangs on the .semget step until the other project releases the semaphore.

See “Semaphores” on page 237 for more information on using this command.

Document # SC23-7681-07

.semput

```
.semput <semaphorename>
```

Releases the semaphore with the name <*semaphore_name*>. See “Semaphores” on page 237 for more information on using this command.

.set

```
.set env <EnvironGroup> "<VariableName>=<DesiredValue>"
```

The .set dot command assigns a value to an environment variable.

Note

Variables set by this command must already exist.

The .set command changes the *master record* for an environment. When the system runs a project, it makes a copy of the project environment from the master record, stores the copy in the job records, and uses that copy as the project default.

The fact that the system keeps a separate job copy of the variables, and that the .set command does not change the job copy, has the following effects:

- When a .set command modifies the master record for an environment, later steps which use the default environment do *not* see the changes, because the system does not refer back to the master record; it uses the job copy for the default environment.
- If you use a .set command to modify an environment, and a later step specifies the same environment the later step will see the changes you made, because the system goes back to the master record for the environment when the step has a specific environment selected. This works even if the step's environment is the same environment as the project default.
- Changes made by a .set command persist after a job is over. Future runs use the values created by previously-run .set commands.

For more information on using this command, see “Changing Environment Variable Values During a Job” on page 111. Also see the similar command “.bset” on page 89.

.sleep

```
.sleep <seconds>
```

The .sleep dot command accepts a number of seconds, and pauses the step for that length of time. Since the Management Console processes this command, no connection to the remote server is created. You can also use a “.sleep 0” command as a platform-independent null command.

Document # SC23-7681-07

.source

Description Use the .source command to add an adaptor for a source code application to a project step. A source code adaptor is a Build Forge object; it is based on the adaptor template for a source code application. The adaptor code for the step is executed when the project runs.

Syntax `.source <adaptor_name> [entry_name]`

The <*adaptor_name*> is required; it is the name assigned to the adaptor in the Management Console. The <*adaptor_name*> case should match the case used in the console.
If your adaptor template has multiple interface functions, specify the one that you want to execute by using the *entry_name* option. The *entry_name* must match the name attribute specified for interface element in your adaptor template. In the following example, the entry name is By Date.
If you are using an adaptor link, the adaptor is called automatically and the first interface function in the adaptor template is executed. To execute a different interface, edit the adaptor template to set the default attribute to true (default="true") on the interface that you want to execute.

Examples
```
.source MyClearCaseAdaptor
```
```
.source MyClearCaseAdaptor "By Date"
```

Notes To create an adaptor or view a list of adaptors, select **Projects** → **Adaptors** .
The adaptor templates provided with the Build Forge product are located in:

`<bf-install>/interface`

.strsub

`.strsub <source> <replacement> file [file ...]`

Use the .strsub command to perform basic string replacement in one or more text files. The system scans the target file(s) for the <*source*> string; where a match is found, the system replaces the <*source*> string with the <*replacement*>. The .strsub command replaces every instance of the string (source) on each and every line in each file.

The .strsub command works across operating systems, without depending on any specific commands being available on the server.

To replace a string _VERSION_ in a file about.c, use a command

```
.strsub _VERSION_ 2.34 about.c
```

You must specify one or more filenames exactly, without using wildcards. For example, a command like the following will fail:

```
.strsub _VERSION_ 2.34 *.txt
```


Document # SC23-7681-07

However, you can use variables in the command, so a command like the following will work if the VERSION and FILENAME variables have been defined in the environment.

```
.strsub _VERSION_ ${VERSION} ${FILENAME}
```

Note

Use spaces to separate parameters in the command. Do not quote the parameters.

The .strsub command is similar to the .edit command; their differences include:

- The .strsub command is faster than .edit when performing replacements in large text files or many files.
- The .edit command can perform regular expression searches and replaces.
- The .edit command replaces the first instance only of the string (search_expression) on each and every line in each file.
- The .strsub command replaces every instance of the string (source) on each and every line in each file.

.test

Description Use the .test command to add an adaptor for a test application to a project step. A test adaptor is a Build Forge object; it is based on the adaptor template for a test application. The adaptor code for the step is executed when the project runs.

Syntax `.test <adaptor_name> [entry_name]`

The *<adaptor_name>* is required; it is the name assigned to the adaptor in the Management Console. The *<adaptor_name>* case should match the case used in the console.
If your adaptor template has multiple interface functions, specify the one that you want to execute by using the *entry_name* option. The *entry_name* must match the name attribute specified for interface element in your adaptor template. In the following example, the entry name is TestFunction.
To execute a different interface, edit the adaptor template to set the default attribute to true (default="true") on the interface that you want to execute.

Examples

```
.test MyTestAdaptor
```

```
.test MyTestAdaptor TestFunction
```

Notes To create an adaptor or view a list of adaptors, select **Projects** → **Adaptors** .
The adaptor templates provided with the Build Forge product are located in:

<bf-install>/interface

Document # SC23-7681-07

.tset

```
.tset env <EnvironGroup> "<VariableName>=<DesiredValue>"
```

The .tset command changes project settings temporarily during a step.

- Use the *env* paramet to specify the environment in which to add or change variable values. You can use .tset to set a variable that does not yet exist. The values set by the .tset command are written to the job record. They do not update the database record for the environment set. Later jobs are not affected by the changes.

The .tset changes take effect in the current step. It takes effect for all commands in the step. It is also in effect for any Inline specified for the step.

Embedding Build Numbers in Project Files

You can use the .strsub command to swap one string for another in files; a common use is to replace a standard token with a system variable such as the $B variable that provides the current job number.

You can use the .strsub dot command to embed build or version numbers in code files. By placing a .strsub command early in your project, a later step can compile files that contain the updated information.

For example, the following steps set up a project to embed build numbers:

1. Add a unique string such as `_BUILD_` to a file in your project. For example, modify a file `README.TXT` and change the version declaration as follows:

   ```
   Application version 5.0.123
   Application version 5.0._BUILD_
   ```

2. An early step in your project should check out the files to be worked on. Add a step after `README.TXT` is checked out which replaces `_BUILD_` with the $B system variable. For the command, use the following:

   ```
   .strsub _BUILD_ $B README.TXT
   ```

3. Run the project and verify that the README.TXT file contains the current job number. For the third run of the project, the README.TXT file should contain this line:

   ```
   Application version 5.0.3
   ```

Enhancements

You can improve this practice in the following ways:

- Use additional environment variables. For example, create variables named $MAJORVERSION and $MINORVERSION and use them as follows:

Document # SC23-7681-07

```
.strsub _MAJORVERSION_ $MAJORVERSION README.TXT
.srsub _MINORVERSION_ $MINORVERSION README.TXT
```

- Update your environment variables when you start a project. By selecting **Jobs → Start** to start your projects, you can see the current environment variables and edit their values before you launch the project. You might include a comment in your jobs, for example, as a variable. Use the Project Action **Must Change** on the comment variable to force users to enter a new value when they run the project.

Changing the Tag During a Job

You can set the value of a tag to a completely new value during the job by using the .retag dot command, which has the following syntax:

```
.retag <new tag value>
```

Here is an example of simple usage:

```
.retag MyProject
```

More complex usage is possible:

```
.retag Job_${B}_${BF_D}
```

This example sets the tag to use the run increment and current date system variables. You can use a command to the server's command interpreter to set the result. To use a command within the dot command, enclose the command in backtick or backquote (` `) characters:

```
.retag `hostname`
```

This example sets the tag to the result of running a hostname command on the server running the step.

Note

Do not mix the backtick form and the standard assignment form of the command.

Changing Environment Variable Values During a Job

You can use the .set and .bset commands to change an environment variable from within a step. These commands change the values of existing environment variables as follows:

- The .set command changes the *master record* for an environment. When the system runs a project, it makes a copy of the project environment from the master record, and uses that copy as the project default. This has the following effects:
  - If a .set command modifies the project environment, later steps which use the default environment do *not* see the changes, because the system does not refer back to the master record.
  - If you use a .set command to modify an environment and a later step explicitly uses the same environment, that step will see the changes you made. The system goes back to the

Document # SC23-7681-07

master record for the environment when the step has a specific environment selected. This works even if the named group is the same as the project default group, so long as the step's environment setting is not “Default.”

- Changes made by a .set command persist after a job is over. Future jobs use the values created by previously-run .set commands.

Use the following basic syntax:

```
.set env <GroupName> "<VariableName>=<DesiredValue>"
```

- Use the .bset command to add or change variable values during job execution. Changes take effect in the step after the step .bset appears in. They are in effect for the remainder of the job.

  ```
  .bset env "<VariableName>=<DesiredValue>"
  ```

  ### Note

  Unlike the .set command, the variable you specify for a .bset command does not have to exist when you set it, so you can use the .bset command to create a new variable during a job. The value of the variable does not persist past the current job.

- Use the .tset command to add or change variable values during job execution. Changes take effect in the current step. They are in effect for any other commands in the step and for any Inline specified for the step.

  ```
  .tset env "<VariableName>=<DesiredValue>"
  ```

  ### Note

  Unlike the .set command, the variable you specify for a .tset command does not have to exist when you set it, so you can use the .bset command to create a new variable during a job. The value of the variable does not persist past the current step.

Setting Multiple Variables

You can set more than one variable at once with these commands by including additional variable and value pairs, separated by spaces, as in the following examples:

```
.set env MyGroup "X=5" "X2=45"
```

```
.bset env "Y=7" "CompilerVersion=4.511"
```

Note

Too many environment variables in a single project can cause Build Forge to hang. To avoid this problem, limit the number of environment variables to fewer than 8000. You can also use .source to set large numbers of environment variables without impacting performance. See “Running Scripts Before a Command With .source” on page 113.

Document # SC23-7681-07

Using Command Output To Set Values

You can generate the value of a variable for a .set or .bset command by sending a command to the server's command interpreter. To use a command within the dot command, enclose the command in backtick characters. For example, the command

```
.set env SetupGroup “PerlVer=`perl --version`”
```

sets the variable PerlVer to the output of the perl --version command.

The variables can only store 256 characters; if more are assigned to a variable, the value is truncated.

By default, the system assigns the entire output of a command in backticks to the variable, but you can use range commands in brackets to select which lines from the command output you want to assign to your variable. The range numbers specify lines from the output using a 0-index (the first line is numbered zero, the second 1, etc.). In the following example,

```
.set env SetupGroup “WindowsIPinfo[0,5-8]=`ipconfig`”
```

the variable WindowsIPinfo receives the first and sixth through ninth lines of the ipconfig command's output.

The following are all valid range modifiers, selecting single lines, groups of lines or combinations:

```
[5]
```

```
[4-6]
```

```
[1,2,5,8-11]
```

The system combines lines without any separation; no spaces or carriage returns are added.

Note

Do not mix the backtick form and the standard assignment form of the command.

Environment Dot Commands

Most dot commands are used in a step's command field, but the .source, .date, and .include commands are used to manipulate environment variables; you use them by providing dot commands as the names or values of variables, as described in the following three sections.

Running Scripts Before a Command With .source

The system provides the ability to execute a script on the server before the system runs the command by defining a special environment variable named .source. This allows you to load a set of environment variables from a source file on the server, or execute a custom preparation command.

To try this feature out:

Document # SC23-7681-07

1. Create a batch file on the system called mybatch.bat that echoes some sentence. Save the batch file to c:\temp.
2. Create a new environment called “Step Variables.”
3. Add a variable called .source with a value of `c:\temp\mybatch.bat`.
4. Edit the Say Hi step of the Hello World project, and set the step's environment to the newly created Step Variables environ.
5. Run the project, and examine the log output for the step.

Notice the additional log data showing that the mybatch.bat file was executed before the step command. Some important notes on .source:

- The path specified cannot have arguments
- On Windows® platforms, the script is invoked via call
- On UNIX® platforms, the script must be in the native shell syntax as it is sourced in the running shell

Storing the Date or Time in a Variable with the .date Command

You can use the .date environment dot command to supply a variable with the current date or time. You use the command as the value of a variable, and the system updates the variable with the result of the .date command when you run a project that uses the variable.

For example, a variable named MONTH with a value .date %B, when included in a project, is supplied to a job during May with the value “May”.

See “.date” on page 90 for more information on using this command, including a list of date format codes.

Using Variables from Other Groups with the .include Command

You can use the .include environment dot command to import variables from a different environment. To use it, create a variable named “.include” in an environment, and set its value to the name of a different environment. When the original environment is used in a project, the referenced environment's variables are set as well.

Note

You can use more than one .include variable in a environment, to include as many other environments as desired. You can use this feature to create nested sets of variables, or to manage a variable in a single location yet use it within many other environments.

Document # SC23-7681-07

Working with Jobs

This section describes how to set up, run, and manage jobs in the Management Console.

About Home

The Home panel provides information about recent jobs and notifications.

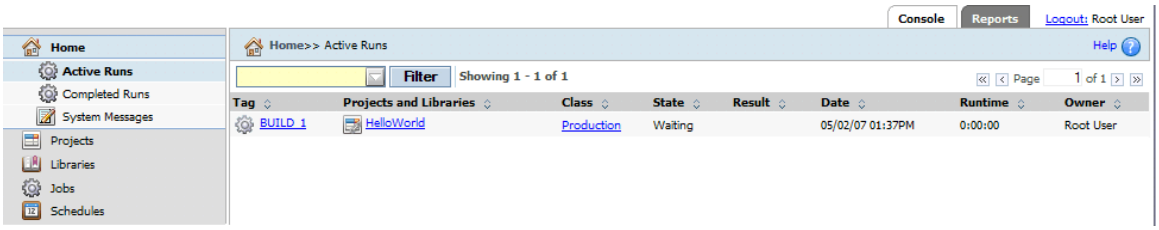
Use the Home module to view your recent or current jobs. You also use it to view notifications and system messages. Select a menu item to view the following:

- **Active Runs** - your currently executing jobs
- **Completed Runs** - your completed jobs
- **System Messages** - the system message log

Use the Jobs module to view all system jobs (if you have sufficient permissions).

Getting Started: For tutorials and how-to topics on using the system, see “Getting Started” on page 2.

The Home Module



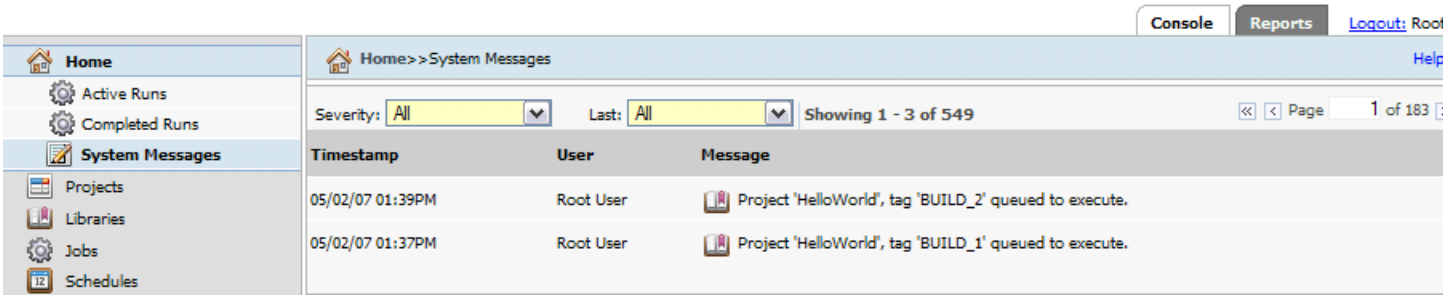
System Messages

The system tracks user and system actions and logs messages about them. To view these messages, select the **Home** module and then click the **System Messages** tab in the lower half of the main content pane.

You can use the **Severity** box to filter the list by the severity of the messages, and you can limit the list to recent messages by changing the value in the **Last** box.

Document # SC23-7681-07

System Messages List in Home Module

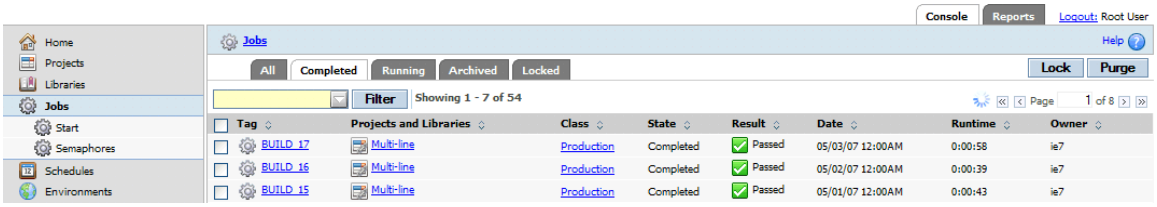


About Jobs

A job is a project that is executing or has finished executing.

Use the Jobs module to launch projects, view the results of earlier runs, and get information on currently-running projects.

The Jobs Module



Jobs, All

The **All** tab in the **Jobs** module lists all projects regardless of job status: completed, running, archived, or locked.

Use the **All** tab to locate a project if its status is unknown. Projects listed on the **All** tab are displayed as follows: currently running jobs are listed first followed by all other jobs in order of date/time completed, with the most recently completed job listed first. Unless the class properties for a project have been set to delete console data, the system retains information such as the job date, logs, and completion status.

Click on a project to display information about the project. The information displayed is dependent upon the job status.

Jobs, Completed

The **Completed** tab in the **Jobs** module lists completed jobs whose logs and data you can view.

Use the **Completed** tab to view jobs which have finished. The system lists jobs in the **Running** tab until they have completed.

Document # SC23-7681-07

Jobs, Running

The **Running** tab in the **Jobs** module lists projects that are currently running on any of the servers known to the Management Console.

Use the **Running** tab to view projects which are in process. The system lists jobs in the **Running** tab until they have completed; after that it lists them in the **Completed** tab.

You can click on a running project to display more information about the project.

Jobs, Archived

The Archived tab in the Jobs module lists information about project runs whose file data has been deleted, but about which the database retains some console data.

Use the Archived tab to view information on jobs which have been purged. Unless the class properties for a project have been set to delete console data, the system retains information such as the job date, logs, and completion status.

Jobs, Locked

The **Locked** tab in the Jobs module lists completed projects which have been locked.

Use the **Locked** tab to view jobs which are in the locked state.

Viewing Step Logs for a Job

You can view and filter the log information for project steps.

1. Click **Jobs** .

   The **Completed** tab is selected. You may click another tab.

2. Click the tag of the job you want to view.

   A list of steps is shown.

3. Click a step name to view its log.

   The step log is shown below the list of steps. Initially all categories are selected.

   If you are viewing a step in a Running job and the step has not completed, you see a partial log and you may not see all categories.

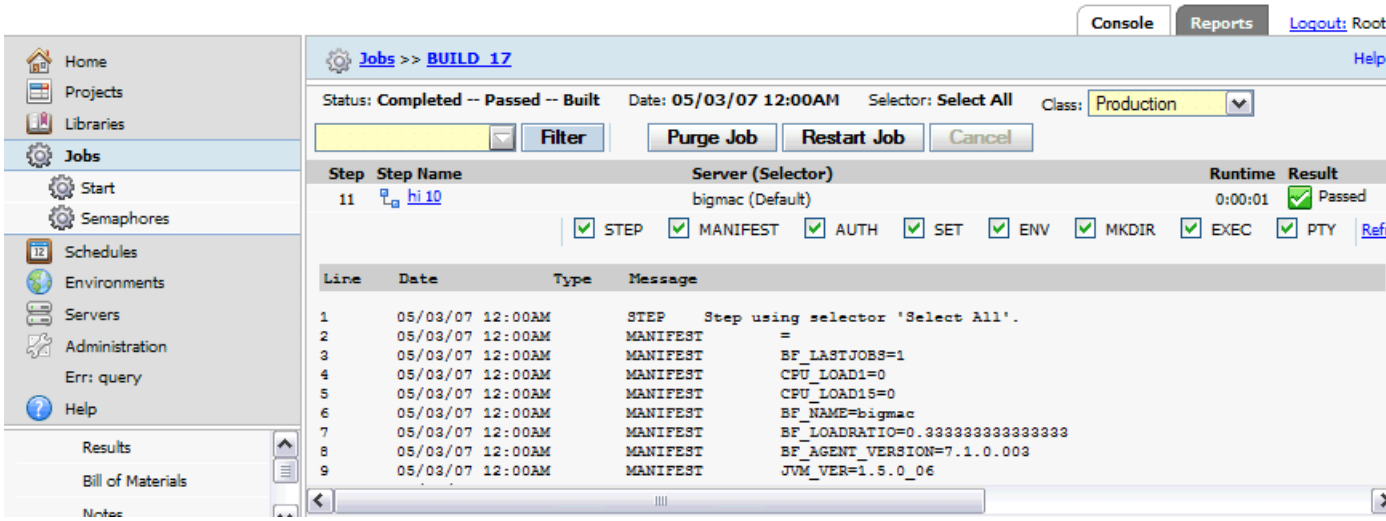
   The log view is a snapshot taken when you clicked the step. To update the log view for a running step, click the build tag (at the top, shown as **Jobs >>** *Tag*), then click the step.

4. Check or uncheck categories, then click **Refresh** to show and hide log output by category.

   The refresh updates the category view, not the log snapshot you see.

Document # SC23-7681-07

Sample Step Log



Locking Jobs

You can lock a project to prevent it from being automatically deleted.

You can lock a job by selecting it in any of the **Jobs** tabs, and then clicking the **Lock** button. To unlock a job, visit the **Locked** tab in the **Jobs** module, and select a locked build; then click the **Unlock** button.

The system will not purge a locked job automatically (as when the class purge properties for the run call for its deletion). A locked job can still be deleted manually.

Use the **Locked** tab to view jobs which are in the locked state. Such projects do not appear in the **Completed** tab.

Restarting Failed Jobs

You can *restart* a job if it fails; this starts a new run which continues from the failure point.

To restart a job, click on the job tag in the list of builds (on the **Jobs** → **Completed** tab). The system displays information about the build, and includes a **Restart** button at the top of the main pane.

1. Click the **Restart** button; the system displays a Restart page.
2. Select options. You can click the **Steps** tab to view the steps in the build. On the Steps tab, select the **Sync Cmds** check box if you want the system to get any updates to the commands in its steps from the project record; if you do not select it, the commands are run exactly as they were when the job was originally started.
3. Click the Restart page's **Restart** button.

Document # SC23-7681-07

A restarted job differs from a new job in the following ways:

- It uses the same tag number as the failed run, and replaces the failed run in the Completed list.
- By default, it starts from the failed step, and does not repeat any of the steps that passed in the previous run. However, you can choose which steps actually run when you restart the job.
- By default, the system supplies the same environment variable values that you supplied on the previous run, with the exception that all environment actions become Sets (see “Variable Actions” on page 33). However, you can change these before restarting the job.
- The system evaluates the success of the job based only on the steps it runs during the restarted job. Failures in the prior run do not affect the status of the restarted job.

If a job fails, the system displays an icon on the **Completed** tab so that you can restart it. Click the icon to display a **Restart** page for the job. A Restart page is similar to a start page, but the system displays the outcomes of the steps from the failed job as colored dots (green for pass, red for fail, yellow for pass with warnings), and steps that ran in the failed job are selected for exclusion from the job.

Semaphores

Semaphores are global signal flags in the system that set up mutually exclusive (mutex) resources. Use them to make some processes wait for other processes to finish.

Use **Jobs → Semaphores** to view job semaphores that are in use. You can also clear semaphores, which may be necessary when a hung or cancelled job fails to release its semaphore.

You implement semaphores through a pair of dot commands: the .semget and .semput commands. Use the .semget command to “grab” a label: after a step gets a label, any other steps (in any project) that try to get the same label must wait until the original requester releases it.

Note

A step that contains a .semget command *waits* until the semaphore is released. If a job fails and leaves its semaphore active, the semaphore must be cleared manually before any job that uses the semaphore can run again.

For example, suppose you have a program that creates a printer driver, and you want the program to be used by only one process at any one time. Within each project that calls the program, set up three steps with the following command lines:

Step	Command Line
Get semaphore	`.semget $BF_PROJECTNAME_PHYS`
Execute driver creator	`printdrivermaker.exe windows`
Release semaphore	`.semput $BF_PROJECTNAME_PHYS`

Document # SC23-7681-07

You can establish semaphores for key resources in your organization, such as a heavily loaded server or a software program with a single-user license. Every step that uses the resource you want to protect should be wrapped with .semget and .semput commands.

Semaphores obey the following rules:

- When used, a semaphore dot command must be the only dot command for the step.
- The system uses the label as-is; avoid using special characters in a label, or trailing whitespace, as you may confuse the label parser.
- Semaphores are global and can be used to synchronize separate projects in addition to thread blocks.
- Semaphores are created the first time they are accessed.
- Semaphores obtained by a project are automatically released when that project terminates.
- If two steps request a semaphore at the same time, there is no guaranteed order as to which step gets the lock.

Clearing Semaphores Manually

The system automatically releases any semaphores created by a project when the project terminates. You can manually release a semaphore that is not released. This can happen if a job terminates abormally.

1. Select **Jobs** → **Semaphores** to display the Semaphores list.
2. Select the semaphore you want to clear.
3. Click **Clear**

Deleting a Job

The following subsections describe several ways to delete a job.

Deleting a Job from the Completed List

You can manually delete jobs from the Completed tab of the **Jobs** module.

You can manually delete one or more jobs from this list. When you do this, the system launches a purge job based on the class of each run; the process is the same as if an automatic deletion had been triggered by the class properties. The system deletes only the data specified for jobs of that class. See “About Classes” on page 62 for more information on class properties.

To delete jobs:

1. Select the **Jobs** → **Completed** page to display the completed run list.
2. Click one or more check boxes on the right end of the table to select builds for deletion.

Document # SC23-7681-07

3. From the list box at the bottom of the list, select the Purge option.

4. Click the Go button.

If the class is set to delete output files but retain console data, then deleting the run from the Completed list deletes the output files and moves the job's entry from the Completed list to the Archived list.

Completely Deleting a Job from the Archive List

Select **Jobs → Archived** to display the archive list. This list displays data about jobs whose files have been deleted. You can delete jobs here just as you would from the **Jobs → Completed** list. Deleting a job from the archive list removes all traces of the job from the database, and drops it from statistics reported by the application.

Automatically Deleting Jobs

The system automatically deletes a job when the class properties for the job determine that it should be deleted. You can use this feature to prevent data from piling up, and to delete groups of jobs.

If you create a schedule for a class, the system only checks for jobs to purge when the schedule activates. See "Scheduling Purges" on page 128 for information on how to schedule purge activity.

To make sure a job is deleted when you want it to be, check the following settings before you launch or schedule the job:

1. Set the deletion properties of one or more of your classes to allow the system to delete the jobs after a certain number of days or after a certain number of jobs have accumulated, or both.

2. Set the **Class** property of the project you are working with to an appropriate class.

If you generate a number of jobs and then want to delete them, you can temporarily change the deletion properties of the relevant class. Or you can select multiple builds on the **Completed** tab and delete them (click the check box next to each job to select it, then click the **Purge** button.

For example, if you generated many Production jobs on the previous day, you would use the following process to delete them:

1. Note the current settings for the Production class.

2. Change the Days property of the Production Class to 1, then click the **Save Class** button. After a 15 minute delay, the system begins deleting jobs that are older than one day.

3. When the jobs are deleted, change the properties of the Production Class back to their original setting.

Document # SC23-7681-07

Working Directories for Jobs

The system constructs working directories for each job, so that each run has a labeled and isolated area in which to work. It makes the working directory names using the values provided for the server path, project name, and tag.

When it executes a command, the system starts the command in the directory specified for the step. By default, that directory is the job's working directory, but you can also specify some other directory relative to the server's **Path** property. The topics in this section describe the path and directory creation processes in more detail.

Note

When it executes a project, the system constructs the project directory (if it does not already exist) and the job directory. It does not construct the server directory (specified in the server's **Path** property) or the step directory (the step's **Dir** property).

Tagging Jobs Dynamically

The system uses *tags* to identify specific jobs of a project, and to construct the name of the job directory in which process activity takes place by default. The system makes the tag for a job from the *tag format* property for the project, which can contain static text as well as numeric *tag variables*.

Note

Do not begin a tag with “BF_”. Tags beginning with“BF_” are reserved for use by the system and cannot be defined by the user.

The default tag format for projects is BUILD_$B, which uses the default tag variable B, an automatically incremented value that is defined by the system for every project. This default tag format results in a stream of build tags as follows:

BUILD_1

BUILD_2

BUILD_3

You are not limited to these tags, however. You can define your own tag variables and set up your own tag formats to produce a variety of tag types. You can also use the .retag command during a job to change the tag to an arbitrary string (see “Changing the Tag During a Job” on page 111).

The current job's tag is available as an environment variable (BF_TAG) defined by the system during a job, so that you can access and use it to label source repositories, or for other tracking or labeling purposes. (For more information on these variables. See “System-Defined Variables (BF_ Variables)” on page 38.)

You can synchronize the tag variables from two projects; this creates a link such that when either one runs, the same tag variable values are used. See “Synchronizing Tags” on page 125 for details.

The topics in this section describe how to set up tag formats and tag variables to produce dynamic tags that reflect the values you want.

Document # SC23-7681-07

Creating or Editing Tag Variables

You can define your own tag variables to include in tag formats. Tag variables take numeric values and can be incremented by the system automatically with each job, if desired.

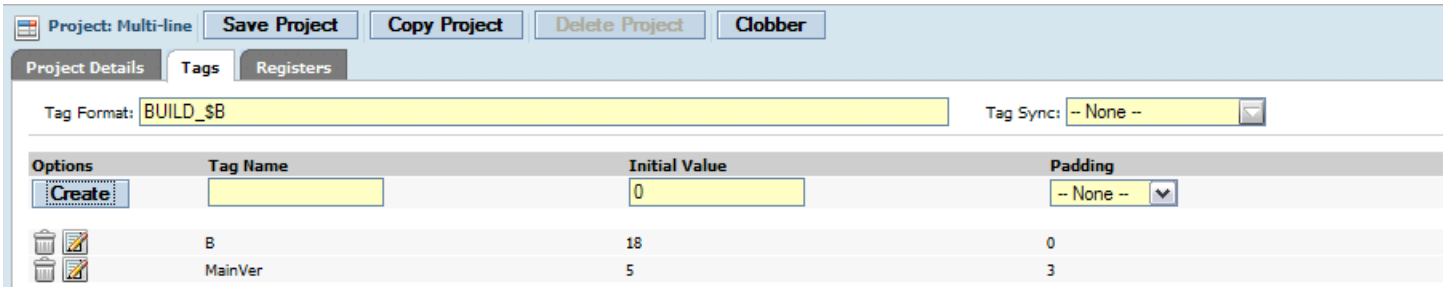
Note

Do not begin a tag with “BF_”. Tags beginning with “BF_” are reserved for use by the system and cannot be defined by the user.

To add new tag variables or edit existing ones for a project, select **Projects** , then click the icon next to the desired project's name. The project properties appear in the lower pane; click the **Tags** tab to display the project's tag variables.

The system displays a list of tag variables for the project.

Tags Tab, Showing Tag Variable Form and Tag Variable List



- To edit a tag variable, click the icon next to its name. The system fills the tag form on the left side of the tab with the tag variable's values and changes the **Create New Tag** button to a **Save Edited Tag** button. Change the values and click the **Save Edited Tag** button to store your changes.
- To delete a tag variable, click the trash can icon next to its name.
- To add a new variable, enter properties for the variable and click the **Create New Tag** button.

Each tag variable has the following properties:

Tag Name	The name of the variable. When you use a tag variable in a tag format, reference its name using the form $<Tag Name>. For example, to create a tag that uses the MainVer and B variables, use a tag format "Build_$MainVer.$B" to get tags like Build_005.1.
Initial Value	Sets the value for the tag variable. If you do not use the **Auto Inc** option, the variable retains this value until you change it.
Padding	If you select a **Padding** value other than None, the system adds leading zeroes to the value of the variable when it is used in a tag if needed to make sure the number of digits equals the **Padding** value. For example, if the variable is

Document # SC23-7681-07

current at 2, and it has a Padding of 3, then the system renders the value as 002. Padding can range from 1 to 8.

Auto Inc — If set to Yes, the system increments the variable's value by 1 for every job of the project.

System-defined Variables for Tags

You can use the following predefined variables in your job tags:

Variable	Value
B	The job number: an integer value that starts at 1 and is incremented with every new job.
BF_D	Date, in the format set by the Tag: Date Format system setting.
BF_J	Day of the year
BF_T	Time, in the format set by the Tag: Time Format system setting.
BF_W	Day of the week (a numeric value, from 0 to 6)

These variables are standard variables populated by the system every time it creates the environment for a step. For example, a tag format value of BUILD_$B.$BF_T produces tags like the following for successive jobs:

BUILD_9.120529

BUILD_10.120533

Editing the Tag Format for a Project

The tag format defines how the system constructs the tag. The tag format consists of plain text and variable references indicated by the $ symbol. Any variables you use in the tag format must be from the list of system-defined tag variables in the preceding section, or must be defined for the project by you before the project is run. Variables that are not defined are treated as static text.

Note

Do not begin a tag with “BF_”. Tags beginning with “BF_” are reserved for use by the system and cannot be defined by the user.

Tag Format is a project property. To edit it, click the Project button to display the list of projects, then click the project name for the project you want to edit. The system displays the list of steps in the project; click the project name at the top of the page to display the project properties.

In your tag format, use a $ symbol to indicate the start of a tag variable. You can include several tag variables if desired. For example, you could define a non-incrementing variable for a project's major revision ($MAJ) and an incrementing variable for its minor revision ($MIN), then have a tag format that reflects the project's version number: Version$MAJ.$MIN. This allows you to manually

Document # SC23-7681-07

control the major version number, but automatically increment the minor version number with every release, producing tags like the following:

Version1.1

Version1.2

Synchronizing Tags

You can synchronize tags across different projects, so that two or more projects increment the same variable, with the project-level Tag Sync property. When you set a Tag Sync property for Project B equal to Project A, you establish a parent-child relationship between Project A (the parent) and Project B (the child).

When you run a project with a Tag Sync property, the system checks to see if any of the tag variables in the child project match variables in the parent project. If the $B variable is used in both tags, for example, and the parent project is on run 7, the child project uses the next value, 8. After run 8, the next run of either project is run 9.

If no variables in the child project's tag format match variables in the parent project's tag format, the Tag Sync property has no effect.

Only the variables in the tag are synchronized, so you can still distinguish between different projects.

For example, if you define two projects as shown in the following table:

Project	Tag Format	Tag Sync
Project A	Project_A_$B	-- None --
Project B	Project_B_$B	Project A

And you then run the projects alternately (Project A, then Project B, then A and B again), your completed jobs list shows the tags as follows (with the last run shown first):

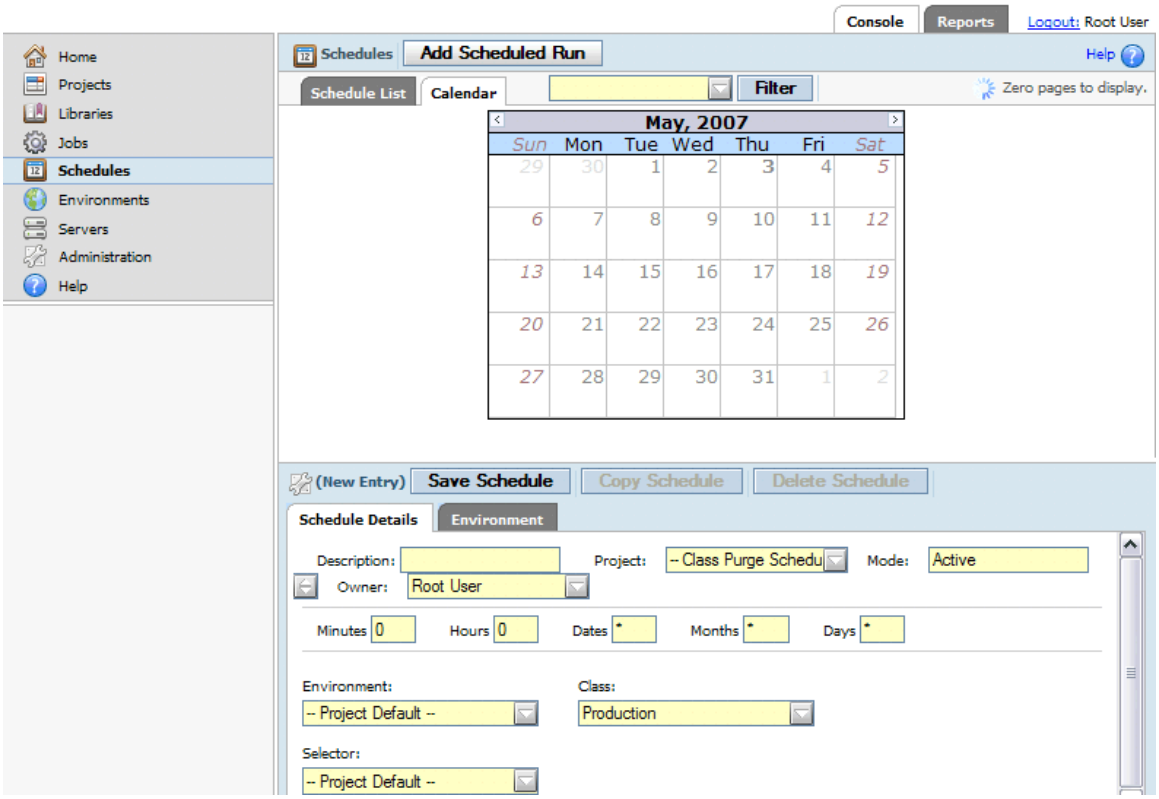
Project	Tag
Project B	Project_B_4
Project A	Project_A_3
Project B	Project_B_2
Project A	Project_A_1

Scheduling Jobs

After you create a project, you can schedule it to run at a future time, or at regular, repeated intervals. For example, you can set up a project to run every hour or every day.

Document # SC23-7681-07

The Schedule Module



Note

If one or more jobs of a project are already running when the schedule activates, the system checks the **Run Limit** property of the project. The system does not launch the scheduled job if the number of running copies of the project equals the **Run Limit** . Set the **Run Limit** value to 1 if you want the system to skip a run if the prior run has not yet completed.

1. Click the **Add Scheduled Run** button.
2. Enter a description for the schedule entry.
3. Select the project (in this case Hello World) from the Project list.
4. You can leave the **Owner** , **Environment** , **Selector** , and **Mode** at their default values for this example, but note the following:
   - The scheduled job runs as if manually launched by the Owner.
   - If you specify an environment here, you can also set starting values just as if you manually started the project.
   - If you specify an environment and/or selector, your selections override the project settings for environment and/or selector for the scheduled run only.

Document # SC23-7681-07

- The Mode field options are Active, Inactive, and Once. When Once is selected, the schedule runs only on the next occasion that matches the time parameters, instead of repeating for every time that matches.

5. Change the Hour field to an asterisk.

6. Change the Minute field to “*/5”. This specifies every 5 minutes. (The schedule parameters work like the UNIX® tool cron.)

7. Click **Save Schedule** to save the schedule entry.

After you add the schedule entry, the system starts to calculate the next runtime for the project; the Next Run column shows “Calculating” until the next scheduled time can be displayed.

When the system has computed the next runtime for the project, it displays it in the Next Run column.

The system displays a dynamic calendar on the **Schedules** page, as well as the form displayed when modifying a schedule. The calendar shows the number of projects scheduled for a given day, for two months (the current and upcoming months) and you can mouse over individual days to see the names and schedule parameters of all the projects scheduled for a given day. If you have more than one project scheduled, the system displays a dropdown box to allow you to filter the calendar display by project.

Use the Refresh link to refresh the page in a few seconds to see the next run time. Wait until the schedule time has arrived and then refresh the **Jobs → Completed** page to see the executed job.

If you end up with many extraneous runs after you experiment with scheduling, you can delete them by changing the properties of the project's class. You can control other features of project types using classes; see the topic on classes for more information.

You can disable a schedule temporarily, or configure it to run once. When you create a schedule, the system displays a green circle next to it to indicate that it is enabled. Click the green circle to change it to a blue one, which indicates the schedule will run once. Click it again to change it to a red circle, indicating the schedule is disabled; while disabled, the project does not run.

You can also use schedules to denote when purges should be performed, so that the system does not try to run purge jobs when the system is otherwise occupied.

Schedule Parameters

This section describes the parameters that you can use to specify when to execute a project. For instructions on creating a schedule, see “Scheduling Jobs” on page 125.

Schedule entries in the system work like UNIX/Linux cron jobs, allowing you to use a series of fields to specify when a project executes.

Field	Description	Range
Minutes	The number of minutes.	0-59
Hours	The hour in the day.	0-23

Document # SC23-7681-07

Field	Description	Range
Dates (Day of Month)	The day of the month.	1-31
Months	The month of the year.	1-12
Days (Weekday)	The day of the week (Sunday = 0).	0-6

The numeric values you enter in the fields may be represented as follows:

- Use an asterisk (*) to denote all valid values in the range. An asterisk can be followed by a forward slash (/) and a step value. For example, a value of */2 in the Hours field executes the project every 2 hours.
- Use a range of numbers separated by a hyphen. For example, 8-11 in the Hours field specifies hours 8, 9, 10 and 11. A range can be followed by a forward slash (/) and a step value. For example, 0-23/2 in the Hours field executes the project every other hour.
- Use a comma-separated list of a set of numbers (or ranges) separated by commas. For example, 1, 2, 3-5, 8.

The project schedule is constructed from the values specified for the fields. In general, the project runs at the time or times that match all the fields as shown in the following examples:

Values Desired schedule	Minutes	Hours	Dates	Months	Days
Run project daily, at 5 p.m.	0	17	*	*	*
Run project weekly, every Monday at 4:30 p.m.	30	16	*	*	1
Run project every half hour on every week day, skip weekends	*/30	*	*	*	1-5
Run project at 12:30 a.m., every other day	30	0	*/2	*	*

However, if the dates and days fields are both non-asterisk values, then the project runs when either of them occurs:

Values Desired schedule	Minutes	Hours	Dates	Months	Days
Run project at 1:01 a.m. on the first day of each month	1	1	1	*	*
Run project at 1:01 a.m. on every Monday in the month	1	1	*	*	1
Run project at 1:01 a.m. on every Monday in the month and on the first day of the month regardless of whether that day is a Monday	1	1	1	*	1

Scheduling Purges

You can control when the system conducts purges of old jobs, by creating schedules for classes of jobs. You create these schedules just as you would create a schedule to launch a project, but

Document # SC23-7681-07

you select the “Class Purge Schedule” option instead of a project. When you do this, the system checks for jobs to purge of the class you select at the time(s) you select in the schedule. For each job that qualifies, the system creates a purge job and puts the job in the waiting queue.

By default, the system checks for jobs that should be purged (based on the class properties which define rules for automatic deletion) at intervals set by the Purge Check Time system setting (which defaults to every 15 minutes). This behavior can lead to purges competing for system resources with ordinary jobs.

If you create a schedule for a class of job, the system only checks for jobs to purge when the schedule activates. If no schedule exists for a particular class, the system uses the default behavior for the jobs of that class. If you want to restrict all purges to specific times, you must create at least one schedule for each class.

To define a purge schedule, create a schedule as usual, but select “Class Purge Schedule” in the Project field. Then select a class in the Class field.

The following example shows how to set up a purge schedule to purge Production-class jobs on Saturdays at noon. With such a schedule in effect, on Saturdays at noon the system checks all the Production jobs, and schedules purge jobs for all the jobs that meet the Production class's deletion criteria.

Working Directories for Jobs

The system constructs working directories for each job, so that each run has a labeled and isolated area in which to work. It makes the working directory names using the values provided for the server path, project name, and tag.

When it executes a command, the system starts the command in the directory specified for the step. By default, that directory is the job's working directory, but you can also specify some other directory relative to the server's **Path** property. The topics in this section describe the path and directory creation processes in more detail.

Note

When it executes a project, the system constructs the project directory (if it does not already exist) and the job directory. It does not construct the server directory (specified in the server's **Path** property) or the step directory (the step's **Dir** property).

Constructing the Working Directory for a Job

The following example shows how the system uses several values to construct a job directory, in a job that occurs on a single server:

System Values	Directory Created
Server's Path field: c:\BuildForge Project name: My Project Tag: Job 5	C:\BuildForge**My_Project\Job_5** The system creates only the **bold** portions of the path. You must create the server directory before running the project, or it will fail.

Document # SC23-7681-07

Note

When it creates a project directory, the system changes characters specified in the **Invalid Relative Dir Characters** system setting into underscore characters. By default, the setting contains a space and a backtick character, so that a project named My Project receives a project directory named My_Project.

If a job occurs on more than one server, the system makes a job directory on each server. Since each step in a project can specify a different server, the system can potentially create many directories. The following example describes a project that uses two servers:

System Values	Directories Created
Project server: ServerA, with **Path** value of c:\BuildForge Third step in project specifies ServerB, with **Path** value of c:\deployments. (All other steps use the default (project) server, ServerA). Project name: My Project Tag: Job_6	On ServerA: C:\BuildForge**My_Project\Job_6**\ On ServerB: c:\deployments**My_Project\Job_6**\ The system creates only the **bold** portion of the path.

In the above example, you can expect any output files from a step to be created by default in the c:\BuildForge\My_Project\Job_6 directory, except for the third step, which uses the c:\deployments\My_Project\job_6 directory.

Constructing Directory Paths for Steps

When the system executes a step, it can start from the directory it constructed for the job, or it can use the **Absolute** option to ignore the project and tag directories.

- When the **Absolute** box is checked, the system constructs the path for the step by adding together the **server path** and the **step's Dir field**. The value in the **Dir** field is a path relative to the server's working directory.

Values for Step	Resulting Path for Command
Server's **Path** field: c:/BuildForge Step's **Dir** field: /bin **Absolute** check box: Checked	C:\BuildForge\bin Use this form to access directories located in the server directory.
Server's **Path** field: c:/BuildForge Step's **Dir** field: / (the default value) **Absolute** check box: Checked	c:\BuildForge
Server's **Path** field: c:/BuildForge Step's **Dir** field: c:/temp	c:\BuildForge\c:\temp (This example will cause an error; the step will fail.)

Document # SC23-7681-07

Values for Step	Resulting Path for Command
Absolute check box: Checked	

Note

You can enter path values with backslashes or forward slashes. The system stores them with backslashes, and changes them to forward slashes as needed on Windows® machines.

- When the **Absolute** box is *not* checked, the system constructs the path for the step by adding together the *server path*, the *project name*, the *tag*, and the step's *Dir field*. The value in the **Dir** field becomes the path relative to the job's working directory.

Values for Step	Resulting Path for Command
Server's **Path** field: c:/BuildForge Project name: My Project Tag: Job_5 Step's **Dir** field: /bin **Absolute** check box: Not checked	C:\BuildForge**My_Project\Job_5**\bin **Bold** portions of the path are constructed by the system if they do not already exist. **Note** When it creates a project directory, the system changes characters specified in the **Invalid Relative Dir Characters** system setting into underscore characters. By default, the setting contains a space and a backtick character, so that a project named “My Project” receives a project directory named “My_Project”.

If the directories specified in a Server's **Path** field or a Step's **Dir** field do not exist, the step fails; the system does not create these directories. The portion of the path specified by the Step's Dir field must be explicitly created during the project by a preceding step.

Typically, steps early in a project check out a tree of directories from source code control, and following steps act on those directories.

When you add new steps, the system remembers the last setting you chose for this check box, and uses that as the default on new steps.

Slashes in Paths

When you enter a path in a **Path** or **Dir** field, the system converts any back slashes to forward slashes. When the system creates the actual path, if the server being used is a Windows® machine, the system converts any forward slashes to back slashes. Therefore you can use either type of slash without concern in your paths.

Document # SC23-7681-07

Note

The system does not modify slashes in the command field of a step. Use back slashes or forward slashes as needed for the target server.

Using the Bill of Materials

The system generates a Bill of Materials (BOM) after each job. The BOM contains information about the steps in the job and the changes to files that resulted from it. The BOM can be provided to consumers of the job, such as your quality assurance department, for help in understanding the contents of a new job. It can serve as an audit solution for your build and release process. With the BOM, you get complete documentation of a job's contents. It can include the results, notes, environments, lists of files, and code changes. This can be used to compare and summarize the state of builds across the enterprise.

The system generates a BOM for each job automatically, but you can use dot commands to cause the system to store additional information about the state of your files before and after the build.

Displaying the Bill of Materials

When you view a completed build (**Jobs** → **Completed**), the system displays the **Steps** tab by default. Click the **BOM** tab to display the Bill of Materials.

Click the + next to any category to expand it. The actual categories you see depend on the project and how your system is configured:

- The Project Steps category appears in every job and displays information about the steps that ran for this job.
- The Source Changes category appears only if your system includes a source code adaptor and the project has a link to the adaptor. See Adaptors and Job Results for details. The format and even the name of the Source Changes category can be changed when you configure your adaptor.
- Baseline and checkpoint sections appear only if your project included .scan commands.

Adding Baselines and Checkpoints with the .scan Command

You can use the .scan command to add more information to the BOM. When the .scan command is executed, the system stores information about the state of the files in the step's working directory.

The command has two forms.

```
.scan baseline
```

Stores a list of all files in the step's working directory tree, with MD5 values for each. The system displays the list in the BOM for the job. You may want to issue this command after performing some setup steps and checking out an appropriate set of files. You can have multiple baseline commands in a project, but each one resets the list to the state of the step's working directory when the .baseline command executes.

Document # SC23-7681-07

`.scan checkpoint`

Stores a list of all new, changed, and deleted files since last .scan baseline or .scan checkpoint in the project, with MD5 values for each file. As with the .scan baseline command, the system displays the list in the BOM. You must issue a .scan baseline command before the first .scan checkpoint command in your project. A .scan checkpoint command that precedes a .scan baseline command will be ignored.

The following example shows how .scan baseline and checkpoint commands work together:

Number	Step	Files after step	BOM data
1	Check out initial files	`config.c` `execute.c`	

2	Baseline	`config.c`
		`execute.c`
			Baseline:
			`config.c`
			`execute.c`

3	Add data file	`config.c`
		`execute.c`
		`data.txt`	

4	Checkpoint 1	`config.c`
		`execute.c`
		`data.txt`
			Checkpoint 1:
			Added `data.txt`

5	Add more data files	`config.c`
		`execute.c`
		`data.txt`
		`data2.txt`
		`data3.txt`	

6	Delete data.txt	`config.c`
		`execute.c`
		`data2.txt`
		`data3.txt`	

7	Checkpoint 2	`config.c`
		`execute.c`
		`data.txt`
		`data2.txt`
			Checkpoint 2:
			Added `data2.txt`, `data3.txt`
			Deleted `data.txt`

Document # SC23-7681-07

Number	Step	Files after step	BOM data
		data3.txt	

.bom

```
.bom addcategory "<Category Name>"
```

```
.bom setcolumn "<Category Name>" "<SectId>" "<Column>" "<Column>" "<Column>"
```

```
.bom data <Category Name>" "<SectId>" "<Column=Value>" "<Column=Value>" "<Column=Value>"
```

The .bom command adds data to the Bill of Materials (BOM) for a build. With it, you can add categories, sections, and rows and columns of data.

Categories — A category is a visible and expandable header within the BOM. It contains a table of data.

Sections — A section is a logical grouping of rows within a category. Section names are not displayed, but the system prints a new set of column headers to mark each section.

The command has three options:

- Use the `addcategory` option to create a new category in the BOM. For example:

  ```
  .bom addcategory "Spaceships"
  ```

  creates a category named Spaceships.

- Use the `setcolumn` option to specify sections and columns of data in a category. You specify a section ID and then column headers. For example:

  ```
  .bom setcolumn "Spaceships" "Section1" "ShipName" "WarpSpeed" "Tonnage"
  ```

  sets up a table of data about spaceships.

- Use the `data` option to specify rows of data. For example:

  ```
  .bom data "Spaceships" "Section1" "ShipName=SpaceShipOne" "WarpSpeed=9" "Tonnage=10000"
  ```

  specifies that SpaceShipOne's warp speed is 9 and its tonnage is 10,000.

As with other dot commands, you can use environment variables in the command. A command like

```
.bom data "Spaceships" "${SECTION}" "ShipName=${NAME}" "WarpSpeed=${SPEED}"
"Tonnage=${TONNAGE}"
```

populates the BOM with data loaded into environment variables by earlier commands.

You can create any number of columns, but the system does not write a line to the BOM until the last column is populated. Also, if you omit a column from a data line, the system uses the value from the previous row. Using the Spaceships example, if you added a line

```
.bom data "Spaceships" "Section1" "ShipName=Freighter" "Tonnage=20000"
```


Document # SC23-7681-07

the system would repeat the WarpSpeed value, giving rows like these:

```
ShipName      WarpSpeed   Tonnage
SpaceShipOne      9        10000
Freighter         9       20000
```

Exporting the BOM as an XML File

This topic describes the syntax, usage, and option descriptions for the bfbomexport command.

Build Forge commands are located in the Build Forge installation directory on Windows and in the <*bf-install*> /Platform directory on UNIX or Linux.

Description

Use the bfbomexport command to export the Bill of Materials (BOM) for a job to an XML file. After collecting BOM information, bfbomexport saves it to the location and file name you specify.

To identify which BOM you want to save to an XML file, you must identify the project and the build for the job.

The bfbomexport command must be executed from the Build Forge installation directory for the Management Console and from the /platform directory on UNIX or Linux.

Syntax

```
bfbomexport [-f filename] [-p projectID | -P projectName] [-b buildID | -t buildTag] [-H]
```

Options

Option	Description
-f filename	An optional path and/or file name. The BOM for the job is saved in XML format. If a file name is not provided, the BOM is written to standard output (stdout). If a path name is not provided, the current working directory is used.
-p projectID	The project ID for the job. (The bfexport command with the -l option lists project IDs.)
-P projectName	The name of the project.
-b buildID	The build ID.
-t buildTag	The build tag name.

Document # SC23-7681-07

Working with Reports

This section describes how to set up and run reports in the Management Console. Quick Report is a separately licensed feature of the system.

About Reports

To display reports about your system, click the **Report** tab at the upper right of the main pane. Reports are provided under choices in the menu in the Reports tab:

- Home
- Performance
- Analyze
- Queries
- Quick Report (a separately licensed feature in Build Forge)

Reports tab

Performance

The **Performance** module shows the last job time for each project and data on the total number of jobs and how many jobs passed, failed, or passed with warnings. Click on a project name in the list to display the project performance detail page, which graphs run times for all the jobs of the project.

Analyze

The Analyze module displays information about the run times and numbers of passing/failing jobs for each project.

Click on an individual project name to display additional information. When you do, the system displays a comparison of the time required to perform each step in different runs and on different servers. The system displays the probability of encountering the longest and shortest run times for each step.

Document # SC23-7681-07

Queries

You can run the following reports from the **Queries** module:

- **Identify the project selectors and step servers for each project:** Click the Run button to display a list of projects and their steps. For each project, the system lists the selector; for each step, the server lists all the servers that the step has ever used in a job.
- **Identify the current manifest for each server:** Choose whether or not to include BF_ properties in the report. This report allows you to compare the manifests for your servers.
- **Build results historic data:** This report allows you to select a range of dates, then displays the jobs in that range, showing the number of passing and failing jobs for each project that had a job in the range. A selected date starts at midnight (00:00) on that date. Therefore, to specify a day's data, select that day for the beginning and the next day for the end.
- **Server and Selector utilization historic data:**Shows server usage over time.
- **Locate a file based on its MD5 value:** You can search through all completed jobs for a file if you have the MD5 value for the file. You can get an MD5 value from the BOM for a job, by running a .scan command.

Quick Report

Quick Report is a licensed option in the Build Forge system. It appears and functions only if you have installed the license key for Quick Report. The installation includes a Tomcat server running on port 8080, so the Management Console should not be set to run on 8080. This section gives you information about Quick Report:

- **Running Reports**: describes how to run an existing report and create a new report.
- **Quick Report Reference**: describes each of the provided report types (shown in the Existing Reports list when you create a new report) and where saved data sources and reports are stored.
- **Setting Up Data Sources**: describes how to configure an existing XML file and import an XML file for use as a data source.

Permissions Needed for Quick Report

Running Quick Report and working with datasources requires the appropriate permissions. The following table lists permissions in the Reports group and lists the Access Groups that have those permissions by default. Permissions are managed in **Administration → Permissions** .

Document # SC23-7681-07

Permissions for Reports

	Build Engineer	Developer	Guest	Operator	Security	System Manager
Read Public Reports	Yes	Yes			Yes	Yes
Save Public Reports	Yes					
Edit Public Reports	Yes					
Delete Public Reports	Yes					
Save Public XML Files	Yes					
Rename Public XML Files	Yes					
Delete Public XML Files	Yes					
Configure Public XML Files into Datasource	Yes					

Running Reports

Running Existing Reports

To run an existing report:

1. Click on the **Wizard** selection in the navigator on the left.
2. Select **Start from an existing report** (or click a report in the list).
3. Click the report to use. Reports are organized under **Public** and **My Reports** headings.
4. Click Run to run the report.

You also have other options to handle the selected report:

- **Edit** : allows you to edit the report in the same way as you create a new report. Remember to save the report when you edit it.
- **Delete** : deletes the report.

Creating a New Report

To create a new report:

1. Click on the **Wizard** selection in the navigator on the left.
2. Select **Create a new Report** . Define the report as follows:
    a. Click a format for the report: Table, Bar Chart, Pie Chart, or Line Chart. When you click on a format, an illustration of it appears to the right of the format list.
    b. **Title** : type a title for the report. It cannot be changed once you specify it here.

Document # SC23-7681-07

c. Click **Next** .

3. **Fields**

   a. Select a report type. Types are organized under **Build Forge** and **Custom** headings. Custom reports come from configuring data sources from XML files (including exported BOM data).

      For Tables, select one or more columns to use as fields in the table.

      For Bar Charts, Pie Charts, and Line Charts: choose columns to use for **X Series Selection** and **Y Series Selection** (X and Y axis of the chart).

   b. Click **Next** .

4. **Group**

   Note: any report that includes a calculated field (such as Total, Percent, etc.) must have a grouping for that field.

   a. Populate the **Group Ordering** list. The report will show groupings and subgroupings as you specify in the list. Grouping is performed in order from the top of the list downward.

      Use the arrow controls to move fields to and from the **Group Ordering** list. Single arrows move a selected item to or from the list. Double arrows move all items to or from the list.

      Use the **Top** , **Up** , **Down** , and **Bottom** buttons to sequence items in the list as you want.

   b. Click **Next** .

5. **Sort**

   a. Populate the **Sort Ordering** list. The report will sort within groups using the fields in the list.

      Use the arrow controls to move fields to and from the **Sort Ordering** list.

      Use arrow control to the right of fields in the **Sort Ordering** to change the sort order (ascending and descending). Clicking on the arrow moves a field in the direction indicated.

   b. Click **Next** .

6. **Filter**

   Build one or more filters. Each filter consists of selections:

   - **Field** : the field in which to filter results based on value.
   - **Operator** : operator on the field.
   - **Value** : either select a field or check the **Text** box and type a value.

   You can **Add more** filters to the list or **Delete** them.

Document # SC23-7681-07

You can limit output by checking **Limit output to** and providing a value for **rows** . This can be helpful when running tests on designs for large reports.

Click **Next** to proceed.

7. **Run**

   The report appears in this panel. If a blank report appears, check that there is data in the data source and that your Filter is set correctly. You have options on how to work with the report:

   - **Save this Report Design** : type a name and click Save. The report will appear in the list of reports.
   - **HTML** : download the report output in HTML format. You are prompted for a location.
   - **Cancel** : exit this report and go back to the Wizard.

Sample Reports

Sample reports are provided in `<bf-install>/samples/reports/`. The files have the extension `.rptdesign`.

To run sample reports:

1. Copy the sample reports to the `public` directory. By default the location is `<bf-install>/Apache/tomcat/reports/public/reports`.
2. Refresh the **Reports** page (or start the console, then click the **Reports** tab, then click **Quick Report**). The sample reports appear in the *Select a report* list in the **Public** folder.
3. Select a report design, then click **Edit** . You can edit the report design at this time (using the tabs) or just save it.
4. Click the **Run** tab.
5. Provide a name (if you want it to be different), then click **Save** .
6. Follow the directions for running an existing report. The sample report you saved appears in the **My Reports** folder.

Important

the report design must be edited and saved as a report before it can be run. If you attempt to run a report design without doing so, database connection error messages are displayed.

Quick Report Reference

Reports Reference

This section describes each type of existing report offered during creation of a new report.

Document # SC23-7681-07

- All fields are available for table reports.
- All fields are available as the **X Series Selection** for charts (independent variable).
- Fields in **bold** are available as the **Y Series Selection** for charts (response or dependant variable).

Analytic — Reports on build performance at the step level using execution time and execution order.

Fields available: **Project Id**, Project Name, **Step Id**, Step Name, Build Tag, **Step Order**, **Step Duration** (minutes).

Build — Reports on build usage and execution performance at the project level.

Fields available: **Build Id**, Server, Build Tag, State, **Run Time**, **Project ID**, Project Name, User ID, User Name, Start Time, Result, **Total Builds**.

Capacity — Reports on servers' overall usage and performance.

Fields available: Project Name, Server, **Run Time**, **Avg Run Time**, **Current Run Time**, Result, Start Time.

Project — Reports on several dimensions of step and environment usage by project, as well as success statistics.

Fields available: Server, **Project Id**, Project Name, Class, **Project Level**, **Project Environment**, Step Name, **Step Order**, **Step Level**, **Step Environment**, Result, # **Passed Steps**, # **Failed Steps**, **Total Steps**.

Quality — Reports on build process quality through success and failure statistics.

Fields available: Result, Run Time, Build Tag, **Project Id**, Project Name, **Total Builds**, Start Time, **Percent Success**, **Percent Failed**.

Resource — Reports on overall and by-server resource consumption as measured by step and build run times.

Fields Available: **Step Duration**, Step Server, Project Name, **Project ID**, Build Tag, **Build Run Time**, Platform, **Step Id**, Build Result, Step Start Time, Build Start Time.

The following table provides a cross-reference of the fields available for use with each type of report. Key:

- **X** - field that can be used only as an independent variable (on the X axis)
- **X Y** - field that can be used both as an independent variable (on the X axis) and a dependant variable (on the Y axis)

Document # SC23-7681-07

Fields Availabile in Reports, by Report Type

	Analytic	Build	Capacity	Project	Quality	Resource
Build fields						
ID		X Y				
Tag	X	X Y			X	X
User ID		X				
User Name		X				
Start Time		X				X
State		X				
Result		X			X	X
Run Time		X Y			X	X Y
Percent Success					X Y	
Percent Failed					X Y	
Total Builds					X Y	
Project fields						
ID	X Y	X Y		X Y	X	X Y
Name	X	X	X	X	X	X
Class				X		
Level				X Y		
Environment				X Y		
Start Time			X			
Run Time			X Y			
Current Run Time			X Y			
Average Run Time			X Y			
Result			X	X		
Step fields						
ID	X Y					X Y
Name	X			X		
Step Environment				X Y		
Step Server						X
Level				X Y		
Order	X Y			X Y		
Start Time						X
Duration	X Y					X Y
# Passed				X Y		

Document # SC23-7681-07

	Analytic	Build	Capacity	Project	Quality	Resource
# Failed				X Y		
Total Steps				X Y		
Other fields						
Server			X	X		
Platform						X

Reports Storage Reference

This section lists where saved reports are stored.

Public Reports `<bf-install>/Apache/tomcat/reports/public` contains three folders:

- `Datasource` contains datasources constructed from XML files.
- `reports` contains saved reports.
- `XML` contains uploaded XML files.

My Reports `<bf-install>/Apache/tomcat/reports/username` contains three folders:

- `Datasource` contains datasources constructed from XML files.
- `reports` contains saved reports.
- `XML` contains uploaded XML files.

Setting Up Data Sources

Reports can be run from XML files configured as data sources, including BOM files exported from the Management Console. The Datasource section allows you to:

- Configure an existing XML file
- Upload a BOM XML file
- Upload a new XML file

Configuring an Existing XML File as a Data Source

A single XML file may be configured as one or more data sources. To configure an existing XML file as a data source, do the following:

1. Click on the **Datasource** selection in the navigator on the left.
2. Select **Configure an existing XML File** (or just click a file in the list).
3. Click on a file in the list, then click the **Configure** button.

Document # SC23-7681-07

4. In the **Configure a new Data Source** panel, do the following:

    a. **Datasource Name** : type a name for the data source.

    b. Select elements from the XML file to use as data in the data source. Select an element, then click **>** to move it to the table on the right.

    c. Configure data fields in the data source. For each row, enter a name for each **Column** and select its data **Type** (Integer, Date, Double, or String).

5. Click **Done** .

The data source is saved. You can access it to create a report in the Quick Report Wizard at the **Fields** step. When you choose a report type, it appears in the list under the **Custom** heading. Standard reports appear first in the list, under the **Build Forge** heading.

Exporting and Uploading a BOM XML File

You can upload a BOM file from one or more selected builds of a selected project. The BOM file is exported from project and build data.

Note

Only BOMs exported using this tool can be used with reports. The XML files from the `.bomexport` command cannot be used for reporting.

To export and upload a BOM XML file:

1. Run a project that exports a BOM file.
2. Click on the **Datasource** selection in the navigator on the left.
3. Select **Use BOM data** , then specify the data to use:
    - ***Name of exported file*** : enter the file name to use.
    - ***Project*** : select a project from the list. Your selection drives the choices in the next step. .
    - ***Select builds to report on*** : click to select a build. Ctrl-click to select multiple builds.
    - ***Choose a destination*** *:* click a folder in Public or My XML Files.
4. Choose whether to configure a data source from the XML file immediately:
    - Click **Upload Only** if you want only to upload the file.
    - Click **Configure** to configure a data source from the XML file now.

If you choose to configure a data source now, do the following in the **Configure a new Data Source** panel:

Document # SC23-7681-07

1. **Datasource Name** : type a name for the data source.
2. Select elements from the XML file to use as data in the data source. Select an element, then click **>** to move it to the table on the right. Click **<** to move elements out of the table.
3. Configure data fields in the data source. For each row, enter a name for each **Column** and select its data **Type** .
4. Click **Done** .

The data source is saved. You can access it to create a report in the Quick Reports Wizard at the **Fields** step. When you choose a report type, it appears in the list under the **Custom** heading. Standard reports appear first in the list, under the **Build Forge** heading.

Uploading a New XML File as a Data Source

To upload an existing XML file:

1. Click on the **Datasource** selection in the navigator on the left.
2. Select **Upload a new XML File** .
3. Click **Browse** to choose the file to upload. You can also type it in. If you type in part of a path and click **Browse** , it starts browsing from that point.
4. Click the destination where you want to store the XML: **Public** or **My XML Files** . Files stored in **Public** can be accessed by other users logged on to the Management Console.
5. Choose whether to configure a data source from the XML file immediatly:
   - Click **Upload Only** if you want only to upload the file.
   - Click **Configure** to configure a data source from the XML file now.

If you choose to configure a data source now, do the following in the **Configure a new Data Source** panel:

1. **Datasource Name** : type a name for the data source.
2. Select elements from the XML file to use as data in the data source. Select an element, then click **>** to move it to the table on the right. Click **<** to move elements out of the table.
3. Configure data fields in the data source. For each row, enter a name for each **Column** and select its data **Type** .
4. Click **Done** .

The data source is saved. You can access it to create a report in the Quick Report Wizard at the **Fields** step. When you choose a report type, it appears in the list under the **Custom** heading. Standard reports appear first in the list, under the **Build Forge** heading.

Document # SC23-7681-07

Working with Utilities

This section describes how to set up and run command-line utilities in the Management Console.

Requirement for Using System Command Line Utilities

When you use command-line utilities such as bfexport or bfimport, the command needs to be able to find the buildforge.conf file in order to access the database, so the command must either be executed from your installation directory or the environment variable BF_CONFIG_FILE must be set to the full path to the buildforge.conf file.

Importing and Exporting Projects

You can export data from the system using a command-line tool or a dot command. You can import data from previously exported files, via the Management Console or a command-line tool.

You can use a command-line tool or the .export command to export projects from your system. For example, you may want to have a step at the end of a project which exports the project data to a file, so that the state of the project is available for backup and to other systems.

Importing Exported Data Objects

You can import a project or other data that was previously exported to an XML file. You can perform an import using the Management Console or a command-line utility.

When you import data objects, the system uses the values of several system settings, especially the **Import with Secure Access** setting, to determine how to apply access groups to the imported data:

- If **Import with Secure Access** is set to Y, then the system ignores access group settings in import files, considering them insecure. All imported objects are assigned to the access group specified in the **Import Secure Default Access Group** setting.
- If **Import with Secure Access** is set to N, then the system honors any access group settings in import files. However, import files do not have to specify access groups. If no access group is specified for an object, then the object is assigned to the access group specified in the **Import Insecure Default Access Group** setting. If that setting is not specified, then the system assigns the object to the most recently created access group which is a default access group for new users.

Importing Projects via the Management Console

To import a file, select **Administration** → **Import** .

Click the **Browse** button to locate your XML file, then select the entities you want to import, and click the **Import** button.

Document # SC23-7681-07

The system offers you a rich set of choices as to which objects you should include in the import.

You can use the **Replace Entities** / **Rename Entities** radio buttons to determine whether the system renames imported objects when they have the same names as existing objects. It does this by adding numbers to them; if you import a Hello World project and one already exists, choosing to rename them creates a new Hello World1 project entry in your project list.

Note

The system does not export server passwords, so you must enter the password for an imported server manually, or apply a new server authorization to it.

Importing Projects via the Command Line

Use the bfimport.exe or bfimport.pl utility (found in your installation directory) to perform command-line imports. You can also use this utility to perform imports through a project. The bfimport command expects the following command line format:

```
bfimport -[L | other options] <filename.xml>
```

Use the -L option to list the data objects in the XML file, or use other options to specify the data objects to be imported. If you do not specify any object types, or if the specified types do not exist in the XML file, the command does not import any data.

Note

The command needs to be able to find the buildforge.conf file in order to access the database, so the command must either be executed from your installation directory or the environment variable BF_CONFIG_FILE must be set to the full path to the `buildforge.conf` file.

The following options are available:

Option	Description

-L	List contents. This option must be used by itself (all the other options can be used in combination with each other). This option lists the objects that are contained in the XML file. For example:
	`c:\Program Files\IBM\Build Forge>bfimport -L CallHelloWorld.xml`
	`Import: 2004:`
	`Project: CallHelloWorld`
	`Project: Hello World`
	`Server: Server3`
	`Class: Production`

-p	Import projects
-I	Include chained projects
-P	Purge logs if replacing projects of the same names
-s	Import servers. Note that the system does not export server passwords, so you must enter the password for an imported server manually.

Document # SC23-7681-07

Option	Description
-e	Import environments
-c	Import classes
-g	
-u	Import users
-T	Import notification templates
-f	Import filters
-r	Replace rather than rename imported items
-A	Import using all users and access groups defined in the import file. Do not map. Create the objects if necessary. **Caution** Security Risk. Restrict access to the Build Forge host to trusted users. This option could be used with a counterfeit export file to introduce new users and access groups or change existing users and access groups.

The following example shows a command to import a project, along with the output produced by the command:

```
c:\Program Files\IBM\Build Forge>bfimport -p CallHelloWorld.xml
Import: 2644: Project import enabled.
Import: 2644:
Import: 2644: Adding project "CallHelloWorld"
Import: 2644: Tag variable "B" import complete.
Import: 2644: Step 1:\"CallHelloWorld\" import complete.
Import: 2644: Project import complete.
```

Exporting Projects from the Command Line

Use the bfexport command to export projects to a named XML file or to standard out (stdout). The export XML file can be used to import projects using the bfimport command or the Management Console (**Administration → > Import**).

You can use bfexport with the -l option to list the names and index numbers of projects in the Management Console database.

Syntax

The bfexport command is located in <*bf-install*>/bfexport.exe on Windows and in <*bf-install*>/Platform/bfexport.pl on UNIX/Linux.

```
bfexport [-l]
```

```
bfexport [-c comment][-f filename] [-g group user] [-s servers][-L LDAP][-n
notification][-help]
<project_name> | <project_index>
```


Document # SC23-7681-07

Usage

On execution, bfexport must be able to find the buildforge.conf file and access the Management Console database:

- Execute bfexport from the directory where buildforge.conf is located (<*bf-install*> on Windows or <*bf-install*>/Platform on UNIX/Linux)
- Execute bfexport from any directory by setting the BF_CONFIG_FILE environment variable for the session to the full directory path of buildforge.conf.

Examples

Use the -f filename option to write output to a file. In the example, the index number (1) is used instead of the project name.

```
bfexport -c "Saving a copy of project before making changes" -f helloworld 1
```

Note

Do not use the redirect output symbol (>) to redirect output to a file; the resultant file includes logging messages and cannot be used as input file for bfimport or the Management Console import.

Option Descriptions

Option	Description
<project_name>	The name of the project. The project name or the project index number is required. List the project name last, after command options.
<project_index>	The index number of the project. The project index number or project name is required. List the project index last, after command options.
-f filename	An optional path and/or file name for bfexport output. If a file name is not provided, bfexport output is saved to stdout. If a path name is not provided, the current working directory is used.
-l list	Lists the projects in the Management Console database by name and index number. This option does not require a project name, is not used with other options, and must immediately follow the bfexport command name, otherwise it is ignored.
- c comment	Adds a comment to the xml output. The comment must be quoted ("my project version 50"). The comment is added to the <buildforge> XML element.
-g group user	Lists the users who are members of the access groups designated to receive notifications. Users and their properties are listed in the <user> XML element.
-s servers	Saves to the XML file the servers defined in the Management Console. Servers and their properties are listed in the <server> XML element, along with any associated <auth> and <collector> information.

Document # SC23-7681-07

Option	Description
-L LDAP configuration	Lists the LDAP domain controllers for the Management Console. LDAP domain controllers and their properties are listed in the <ldap> XML element.
-n notifications	Lists the user-created notification templates assigned to the project. The notification templates and their properties are listed in the <mail-template> XML element.
-C collectors	Saves to the XML file the collectors assigned to the servers in the Management Console. Collectors and their properties are listed in the <collector> XML element.

Example

```
C:\Program Files\IBM\Build Forge>bfexport -l
1: Hello World
2: Production Run
3: Test Run
bfexport -c "Saving a copy before making major changes" 1 > helloworld.xml
```

The above command line writes the project data to a file helloworld.xml in the current directory. The data includes the comment as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<buildforge schema="3.5" comment="Saving a copy before making major changes.>
<project primary="1" name="Hello World" active="Y" exclusive="0" maxthread="0" tagsync="0"
 access="Guide">
```


Document # SC23-7681-07

Administering the Management Console

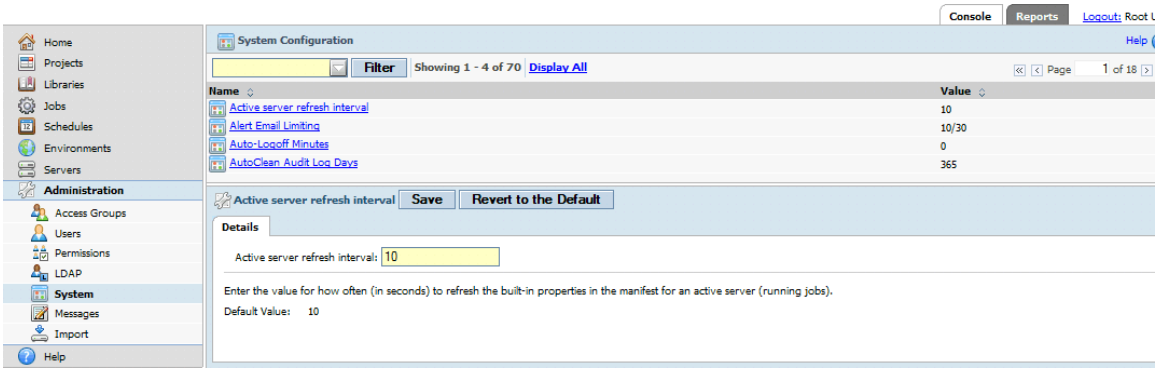
This section describes administrative operations for the Build Forge® system.

About Administration

Use the Administration module to manage configurations and preferences.

Within the Administration module you can work with users, security privileges, notification settings, collectors, and system settings to configure your Management Console.

Administration Module, System Settings



Security Overview

The system manages users in its database, and allows you to control the privileges of users (through the groups you assign them to). You assign privileges to groups, then make each user a member of appropriate groups.

This is a role-based system: each group represents a role a user can have in your organization, and roles have privileges. A user's privileges are the sum of the groups the user belongs to. You cannot assign privileges to individual users directly, only to groups.

The system also uses access groups for notification. When you ask the system to send notification messages, the target of the messages must be an access group. See “Setting Up Notification” on page 64 for more information.

Security privileges, or *permissions*, define what a group can do and/or see. They can serve as a filter on the group's experience of the system. For example, a user who is a member of the Guest group (and no other groups) sees only Projects which have the Guest group assigned as their Access property. That user can only launch projects with Guest access. If the user was also a member of the Developer group, he would see all the projects whose Access properties were either Guest or Developer.

Note

You can use an existing LDAP database for user authentication, instead of the database. When you do this, instead of defining users in the system, you allow some or all of the users from your LDAP database to access the system. You can also map access groups to LDAP groups. For details on setting up LDAP, see “LDAP and Active Directory Integration” on page 159.

The activities and resources that you can control with access groups are Permissions, Servers, Projects, Steps, and Access Groups.

- To extend access to a resource (Server, Project, or Step) to a particular group, select that resource and change its Access field to that group's name. For example, to give the Developer group access to a particular server named Win234, set the server's Access property to Developer. **NOTE:** A user who is not a member of a server or project's Access Group does not see that object listed on the Server or Project list pages. A user who is not a member of a step's Access Group can see the step in the list for the project, but cannot edit it or run it; if the user runs the project that contains the step, the system skips the steps the user does not have access to.
- To allow members of one access group to edit another access group, set one group as the Control Group of the other. For example, to allow Group A members to add members to Group B, make Group A the Control Group for Group B.
- To extend a global privilege to a group, use the **Administration** → **Permissions** page to enable a particular permission for that group.

This flexible model allows you to securely give one privilege (such as the ability to run builds) to some types of users, while restricting others (such as the right to edit projects or use certain servers).

Security Example: Giving A Group the Ability to Run Jobs

You can use the security features to extend the ability to run certain Jobs to one of your Access Groups. For example, you might have a group of device driver programmers and you want to allow them to run jobs that are relevant to their work, without cluttering their view of the system with many other jobs, and without allowing them to edit the jobs you create. To create this scenario:

- Create an access group for this role in your organization ("DeviceDriverDevs").
- Assign the new access group as the **Access** property of all the projects you want the users to be able to run.
- Make sure the steps of the projects have appropriate **Access** properties as well. Any steps that the users do not have access to are skipped when the job runs.
- Assign the permission Execute Builds to the group.
- Make all the users who need to launch these builds to the new DeviceDriverDevs group. You may also need to make administrators of the system members of the group as well; when you

change the Access property of the projects, users who are not members of the DeviceDriverDevs group lose the ability to view, execute, or edit the project.

Note that users can be members of many groups, and permissions are cumulative. You could have a group for another project team ("PlatformDevs") and a user who was a member of both groups would be able to view and launch projects that had either group set as the Access property.

Team and Project Security Plan

If you have many users who work on different projects, the following general plan provides you with the ability to manage them so that individual users get the permissions they need but only see the projects and other resources that they need to interact with:

- Create role-based access groups for the various activities people perform. For example, create Build Manager and Developer groups. Assign permissions to these groups as appropriate for their jobs. Build Managers might have most of the available permissions, while Developers might have only permissions related to executing jobs.
- Create additional groups for each cross-functional team in your organization. You might have an IDE team, a PrinterDriver team, and others.
- Set the Access properties of projects, servers, and other resources to team groups. All the projects that are relevant to the PrinterDriver team should have the PrinterDriver access group as their access properties.
- When you add a user to the system, assign a user to all the appropriate access groups. All users must be assigned to at least one role group and at least one team group.

If you follow these guidelines, users see only the projects that are relevant to them, and have permissions appropriate to their roles in those projects. Also, you can easily change user permissions as their jobs within your organization change.

Managing Access Properties

When you assign access properties to data objects such as projects, servers, or steps keep in mind that a user can only assign a project or server to an access group to which they are a member.

If you are not a member of the Admin group, you cannot assign a project to that group. The list of access groups is restricted to those to which you are a member.

For example, steps inherit their access group properties from their parent project. The step creator can change the access group property for a step so that it has a different access group property than the project. Users who are not members of the access group specified for the step cannot execute the step. This allows you to prevent users from running specific steps in a project.

Access Groups

An access group is a collection of users that the system uses to control permissions and to handle notifications.

Document # SC23-7681-07

Use the **Administration → Access Groups** page to create new access groups, to add/remove users, and to modify group properties. The page displays a list of existing access groups. Click the name of a group to select it and display its properties in the lower pane.

- To create a new group, click the **Add Group** button to clear the fields in the lower pane (if needed). Then give the group a **Name** and select an **Owner** group (a group to control access to the new group; to edit a group, a user must be a member of the **Owner** group).

  If you are using LDAP authentication, fill out the LDAP Group DNs field to tell the system which LDAP groups to map to your group. For example, you might map the Developer group to an LDAP group named SoftwareEngineers, and then assign permissions to the Developer group to provide the type of access you want your software engineers to have.

  - In the LDAP Group DNs field, list the Distinguished Names of all the LDAP groups whose members should receive the Management Console security privileges associated with this access group.
  - You can map multiple LDAP groups to any access group. You can use the asterisk (*) character in this property to give all LDAP users membership in this access group. You can list multiple LDAP groups and separate them with semicolons.

- To delete an access group, select the group, then click **Delete** .

  ## Note

  You cannot delete access groups that are assigned as an **Access** property elsewhere. For example, if you create an access group, then set the **Access** property of a project to use it, you cannot delete the group. You must first edit the project to use another access group.

- To add or remove users, select the group, then click the **Users** tab. The system displays a list of nonmembers on the left and members on the right. Select users and use the **Add** and **Remove** buttons to move them from one list to another.

- You can nest groups, by adding a group as a subgroup of another group. When you do this, all the notifications and permissions that apply to the containing group apply to all the users in member groups as well. To make one group a subgroup of another, select the desired parent group, then click the **Subgroups** tab in the lower pane. Select the groups you want to make into subgroups, and click the **Add** button.

  You can even recursively nest groups, for example, add a parent to a child so that group A contains group B which contains group A. If you do this, the system treats all members of group A as members of group B, and vice versa.

- To assign permissions to a group, choose a permission and make sure the desired group is listed as having that permission. Select **Administration → Permissions** , select a permission from the list, and then work with the lower pane. Groups on the left side lack the permission; groups on the right have the permission. To give a group a permission, select it in the group on the left and then click the **Add** button.

Document # SC23-7681-07

Creating and Editing Users

You can create users and assign properties to them via **Administration → Users.** You can also connect your system to an LDAP/Active Directory database to get user information. You manage user security permissions by assigning users to groups, so you must create some users in order to test security features.

Select **Administration → Users** to display a list of current users, with a user form below it. The system displays the Name, Login, Email, Limit, Activity (elapsed time since last user activity), and Time Zone of each user.

- To edit a user, click the user's name and edit the properties in the user form, then click **Save** .
- To create a new user, start entering properties in the user form when no user is selected. If a user is selected, click the **Add User** button to clear the form. Click **Save** when you are done editing user information.
- To log out a user, click the user's name and then click **Logout User** .
- To log in as a user without using their password, first log in as root. Click the user's name, then click **Switch to User** .
- To free up a fixed license seat, first log in as root. Click the user's name, then click **Purge Seat** . The console removes the user from the list of IDs counted toward the set of fixed licenses. The user is also logged off if he or she is logged in. For fixed licenses, the console counts the number of users who have ever logged in. Once the limit is reached, no new users can obtain licenses. Existing users must be deleted or purged in order to make room for another user. Purging a seat does not delete the user from the console. If the user logs back in, the fixed license count is increased. If used on a floating-license user, **Purge Seat** has the same effect as **Logout User** .
- To delete a user account, click the user's name and then click **Delete**.

  If the Delete button is disabled, a scheduled job is owned by the user account and the user account cannot be deleted. If you want to delete a user account that has scheduled jobs, you must first delete the scheduled jobs.

The user record sets default properties for the user's experience of the system and controls the user's login name, password, and password expiration. The data for a user record can be entered into the system through the Management Console, or derived from an LDAP/Active Directory database.

Note

When you edit a user whose record is derived from an LDAP database, many of the fields on the User page are disabled. You must change these properties in the source database.

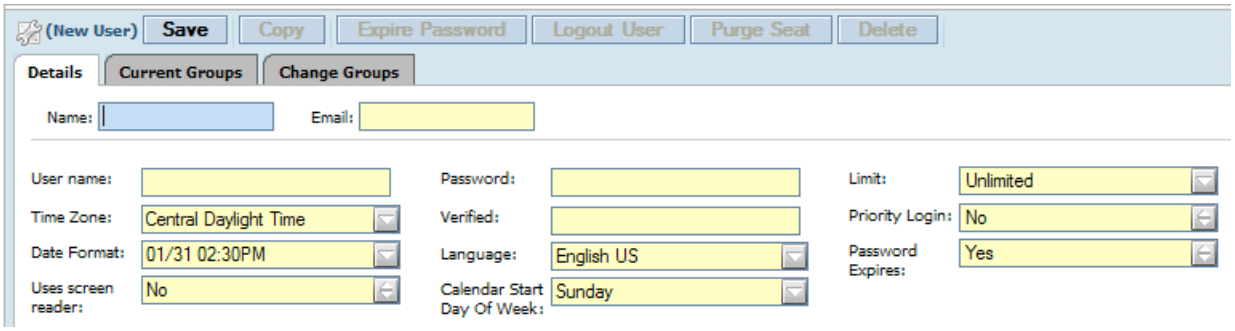
To add a new user, click the Add User button, edit the form, then click the Update button.

When you display a user record, three tabs are available:

Document # SC23-7681-07

- **Details** : Use this tab to edit most of the user properties. The available properties are described below.
- **Current Groups** : Displays the access groups the user is a member of, either directly or through a direct group being a member of another group.
- **Change Groups** : Displays the groups the user is a direct member of, and allows you to add the user to groups or remove the user from them.

User Details Form



For each user, you can set these properties on the **Details** tab:

Name	The display name and label for the user.
Email	The e-mail address where the system can send e-mail notifications for this user.
User Name	The name used to log on to the Management Console.
Password	The password used to log on to the Management Console. The field is not displayed for the user currently logged on. Use this field to enter a new password or change the existing one. Enter the same password in the **Verified** field.
Limit	Sets the maximum number of jobs the user can launch in a day. When the limit is reached, the system displays messages indicating that the user's run quota has been exceeded. If the value is 0, the system allows the user to run any number of builds.
Time Zone	The user's time zone. The system uses the time zone of the root user as the default time zone for all times posted. By default, in-system users and LDAP users are assigned the same time zone as the root user. Users can edit the time zone assigned to them.
Verified	Repeat the password here to verify it.

Document # SC23-7681-07

Priority Login	If this option is checked, the user becomes a priority user. A priority user can always log in to the system; if there are no more available user licenses, the system logs out the user with the oldest session to make room for the priority user. The root user is always a priority user.
Date Format	Selects the user's preferred display format.
Language	Selects the user's language.
Password Expires	If this option is checked, then the user's password expires after a number of days have elapsed, as specified in the **Password Expiration Days** system setting.
Uses screen reader	If set to Yes, the interface is enabled to support screen reader features for vision impaired users such as dynamic highlighting and focusing.
Calendar Start Day of Week	Select the day of the week that the Schedule calendar displays first. The default is Sunday.

Root User

The root user (the user whose login name is **root**) has special characteristics within the system:

- **Created upon installation**: the root user is the only default user created by the installation program. The default password is **root** (you should change the password immediately after installation).
- **No license required**: the root user does not consume a user license. No matter how many users are logged in, you can always log in as the root user. (When someone logs in as root, any other user already logged in as root is logged off.)
- **System time zone**: the root user's time zone is the default time zone of the Management Console. The time zone for other users, both in-system and LDAP users, is taken from the root user's time zone by default. Users can set their own time zone after logging in once. All times and logs reported in the system are expressed in the user's time zone.
- **All permissions**: the root user has all available permissions and can edit other user's properties. You cannot remove any access privileges from the root user. Although the root user is not a member of any access groups, the root user can view, edit, or use any data objects in the system.
- **Priority**: The root user is always a priority user.
- **Log in as any user**: The root user can log in as a user without using a password by clicking **Switch to User** on the **Administration → Users → <UserName>** page.
- **Logging out current users**: the root user can log out users by clicking the **Logout User** button on the **Administration → Users → <UserName>** page.

Document # SC23-7681-07

- By default, LDAP users are assigned the same time zone as the root user. However, they can edit the time zone assigned to them after they log in once. The system remembers the new preference.

API Users

The API uses a standard console login to access the system. A script that uses the API to log in to the Management Console must have a valid console username and password. IBM recommends that you set up a special user for the API, as otherwise a script invalidates the session of a user with the same login name who is logged in to the console when the script executes.

Permissions

Permissions define what a user can do within the system. You assign permissions to access groups; you do not assign them directly to users.

To work with permissions, select **Administration → Permissions** .

To *assign permissions* to a group, choose a permission and make sure the desired group is listed as having that permission. Select **Administration → Permissions** , select a permission from the list, and then work with the lower pane. Groups on the left side lack the permission; groups on the right have the permission. To give a group a permission, select it in the group on the left and then click the **Add** button.

Permissions Exercise

In this example, you give the user Jane Doe exclusive rights to add and edit servers by giving those permissions to a new Access Group and assigning her to that group.

1. Login as root.
2. Make a new Access Group called Server Admin.
3. Add Jane Doe to the new Access Group.
4. Go to **Administration → Permissions** .
5. Scroll to and click the **Add New Servers** permission.
6. In the Details tab, use the **Add** and **Remove** buttons to make Server Admin the only Access Group that has this permission.
7. Click **Update** . Now only Jane Doe has permission to add servers to the system.

Note

The root user always has all permissions. You cannot remove any access privileges from the root user.

Document # SC23-7681-07

Notice that only Jane can add a server. However, she does not have the ability to enter the server's login information. This can be done by adding the Server Admin group to the **Edit Server Authentication** permission.

LDAP and Active Directory Integration

You can use the system with an LDAP server or Active Directory server so that your users can use the same login names and passwords they use elsewhere in your organization, and you can avoid maintaining an additional set of user records. When you use LDAP, you do not have to manually create users in the system, but you retain the ability to create users who exist only in the system.

The system does not use LDAP authentication until you create at least one LDAP domain.

The system remembers LDAP users after they log in once. LDAP users appear on user lists only after they have logged in at least once.

When a user logs in while the system is configured to use LDAP, the LDAP adaptor uses an administrative account to log in to the LDAP server and search for the username supplied by the user. On finding the user in the LDAP database, the system then rebinds to the LDAP database using the username and password supplied by the user; if this succeeds, the user is allowed into the system and the system stores a record of the user.

A user record whose information is derived from an LDAP database has its **User Name** , **Password** , **Login** , **Confirm** , and **Email** fields disabled. Other properties, such as the time zone and access groups, can be changed within the system and the system remembers those values. (The system assigns LDAP users to the *root user's time zone* on first login, since it does not get time zone information from LDAP.)

Note

If you want to change values that are derived from LDAP, you can either create the user record in the system before the user logs in (since the system does not replace an existing user record if the system already has a user with the same login name as an LDAP user), or you can delete the user derived from LDAP and then create a new user with the same login name. Deleting the user deletes all Build Forge properties associated with the user.

Note

You can map LDAP groups to access groups by setting the **LDAP Group DNs** property for an access group to reference the distinguished names of one or more LDAP groups.

Note

Group mapping is performed only once, upon initialization. Changes to LDAP groups are not checked for once Build Forge is initialized.

Document # SC23-7681-07

Creating LDAP Domains

You can create as many LDAP domains as you like. Each domain has a full set of configuration parameters. To add a domain, select **Administration → LDAP** , then click **Add LDAP Domain** . Enter values for the new domain's parameters, then click **Save** .

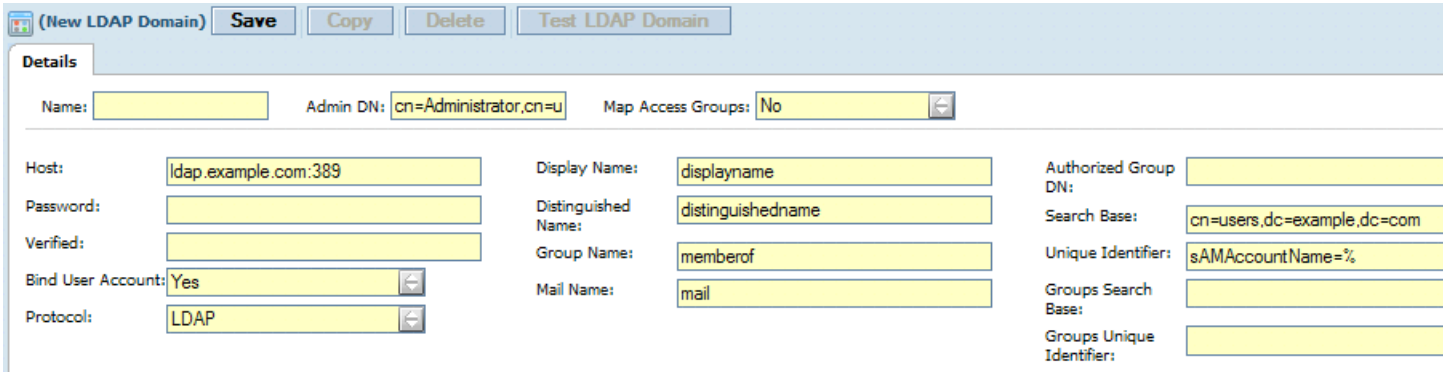
Note

If you have only one domain, the system automatically uses it for authentication. If you create more than one, the system provides users with a domain drop-down list on the login page.

Enter a name for the new domain, to identify the domain within the Management Console. The domain name is not used outside the console.

When you add a domain, the system creates a set of parameters for the new domain and applies the default LDAP/Active Directory configuration values to them. The default parameters are designed to work with a standard Microsoft® Active Directory server.

LDAP Domain Configuration Parameters



Support for LDAP Traffic over SSL (LDAPS)

You can use a directory service (LDAP server) in your network to authenticate Build Forge users.

If your LDAP server is enabled for LDAP traffic over SSL (LDAPS), first configure an LDAP domain in Build Forge, then complete the following set up to allow Build Forge to access and use the directory service on the LDAPS server.

For Management Consoles running on Windows, do the following:

1. Install PHP 5.2.1.
2. Create a file called ldap.conf.
3. Add the following line to the ldap.conf file: `TLS_REQCERT never`.
4. Save the ldap.conf file to C:\openldap\sysconf\ldap.conf.

Document # SC23-7681-07

For Management Consoles running on Unix, do the following:

1. Locate the ldap.conf file (/etc/ldap/ldap.conf).
2. Add the following line to the ldap.conf file: `TLS_REQCERT never`.
3. Save the ldap.conf file.

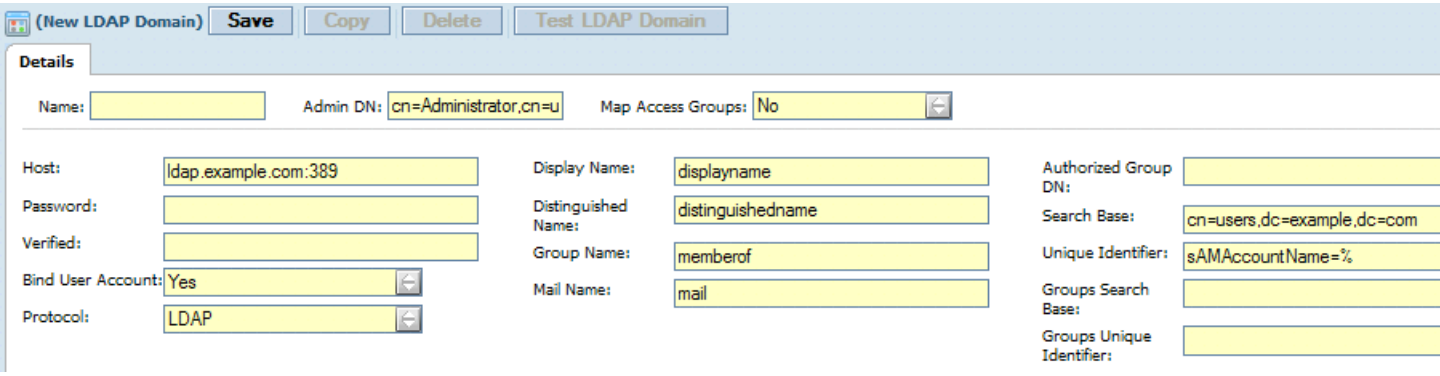
Turning Off LDAP/Active Directory Support

If you want to stop your Management Console from checking logins against an LDAP/Active Directory server, select **Administration → LDAP** to display the list of LDAP domains, then use the trash can icon next to each domain to delete it. When no domains exist, the system uses only built-in users (those stored in its own database) to authenticate logins.

LDAP Domain Parameters

After you create a new domain, you can select **Administration → LDAP → <Domain Name>** to display the parameter page for a specific domain. The system assigns default values to some parameters (and leaves others blank) when you create the domain.

LDAP Domain Configuration Parameters



Edit the values for any of the fields, then click **Save** to add the new values to the database. The following topics describe each category of parameters.

The first row of parameters covers basic properties:

Name is the system name for the LDAP domain. It need not be the same as the host name. This is an identifier used only by the system to store a set of LDAP parameters. When you use an IDE plug-in to connect to the system, use this name when configuring your login to use a particular domain. Also, when a user logs in to the system, a **Domain** field appears on the login page if any domains exist; the field lists the domains according to their **Name** values.

Admin DN is the account to use when searching the LDAP server database. If your server allows an anonymous bind for searching the database, leave the Admin DN field blank. Some LDAP servers require an administrative bind in order to search the database. This setting allows you to specify the DN of the administrator account, as shown in the following example. Use a comma-separated list to specify one or more accounts.

Document # SC23-7681-07

```
cn=Administrator,cn=users,dc=example,dc=com
```

Specify the password for the Admin DN account in the Password and Verify Password fields.

Map Access Groups selects whether or not the system tries to map group information from the LDAP server to access groups in the Management Console to determine what groups the user should be a member of. The default value is No.

- Set this value to **Yes** if you want to get access group information from your LDAP database. If you set it to yes, the system gets access group information from the LDAP database every time an LDAP user logs in, meaning that any changes made to the user's group memberships in the Management Console are overwritten. When you set this value to Yes, you must map LDAP groups to Management Console access groups by setting the **LDAP Group DNs** property for each access group.
- If you are using LDAP and this setting is set to **No**, then you must assign LDAP users to groups within the Management Console after they have logged in to the console at least once.

After the first row, the parameters are organized in columns. The following paragraphs describe them, column by column:

Host is the host name (and optionally, port) of the LDAP or Microsoft® Active Directory server. Required.

Examples:

```
ldapserver.mycompanyname.com
```

```
ldap.mycompany.com:9000
```

Password stores the password for the Admin DN account.

Verified repeats the password for verification purposes.

Bind User Account determines whether the system attempts to validate the user's credentials:

- When this parameter is set to **yes**, the system uses the credentials supplied at login to re-bind with the LDAP server and validate the credentials.
- When this parameter is set to **no**, the system assumes that the username passed in has already been validated (as in a single-signon environment) or that some external password validation mechanism has been supplied to the Management Console (contact support for help with implementing such a system).

Protocol identifies the protocol Build Forge uses to read and write data from the directory service for the purpose of authenticating Build Forge users. The default is LDAP. If you need to support LDAP traffic over SSL (or LDAPS), select LDAPS, and complete additional setup to allow Build Forge to access and use the LDAPS-enabled directory service. See .

Display Name identifies the keyname in the LDAP schema that holds the user's full name.

Distinguished Name identifies the keyname in the LDAP schema that holds the Distinguished Name for a user account.

Document # SC23-7681-07

Mail Name identifies the keyname in the LDAP schema that holds the user's email address.

Group Name identifies the keyname in the LDAP schema that holds the list of groups the user is a member of. This group list drives the authentication permissions and access permissions.

Authorized Group DN defines the distinguished name of an LDAP group that should be allowed to log in to the Management Console. Leave this setting blank to allow *any* valid LDAP user to log in to the console. When it is set to the LDAP DN string for a group in the LDAP database, then only members of that group are allowed to log in to the console.

Search Base defines the search string used to query LDAP records for users. Required.

Example:

```
cn=users,dc=buildforge,dc=com
```

Unique Identifier identifies the field in the LDAP database that should be compared against the user login. Use a % character as a placeholder for the login name entered by the user. Thus, this filter maps the user's login name to a field in the LDAP database. Required.

Example:

```
(sAMAccountName=%)
```

In many LDAP databases, group membership information is part of each user record. But if your LDAP database stores group membership in a separate table, the system can perform a separate query to get that information. If you need such a query, configure the following two parameters:

Groups Search Base defines the search string used to query LDAP records for group data.

Example:

```
cn=groups,dc=buildforge,dc=com
```

Groups Unique Identifier identifies the field in the LDAP database that should be compared against the user login to uniquely identify the user, but in this case it is used to filter the user's data from the group table(s). The filter can use any of the data fields retrieved from the user's account as a key into the groups table. Use the %fieldname% syntax to identify a field.

For example, if your groups table has each user's sAMAccountname field as a key, you might use the following as your filter:

```
sAMAccountName=%sAMAccountname%
```

Testing LDAP Configuration for a Domain

To verify that your LDAP configuration is working, select **Administration → LDAP** and then select a domain from the list. Then click the **Test LDAP Domain** button. The system queries the LDAP server, verifying that it can make a connection using the configuration information you have supplied.

Document # SC23-7681-07

System Configuration Settings

You can use a variety of settings to configure your Management Console. These settings are found on the **Administration → System** page.

When you click **Administration → System** , the system displays a list of settings. Click on the name of a system setting to display an edit form for the setting.

Note

For system settings that take numeric values, the Management Console accepts any value comprised of one or more integers (0 through 9). Numeric grouping characters are not supported; for example, commas (,), decimals (.), and other noninteger separators.

The form includes the following buttons:

- **Save:** Saves any changes you make to the setting's value.
- **Revert to the Default:** Resets the setting to its default value.

The following table describes the available settings.

Setting	Description
Active server refresh interval	Sets the time, in seconds, between updates of Built-in and Set Value properties for server manifests on active servers: servers that are currently running projects.
Alert Email Limiting	Sets the maximum number of alert e-mails the system sends over some period of time. This value should be expressed as <number of e-mails>/<time in minutes>. For example, the value 10/60 sets the maximum to 10 messages per hour. The default value of 0/0 is interpreted by the system as no limit on messages. The system maintains a counter of messages during the limit period, and stops sending messages when it reaches the limit, until the end of the limit period, when it resets the counter.
Apply inlined steps container environment	Default: No. If Yes, applies the environment of the project or library that contains an inlined step.
Apply server environment last	Default: No. If Yes, applies the server environment for the step last. The server environment is applied after the step environment or project environment, if these environments are specified.
Auto-Logoff Minutes	The system can automatically log off users who are idle. This setting specifies the number of minutes of idle time that must pass before the system logs off a user. When the setting is 0, the system does not attempt to automatically log off users.
AutoClean Audit Log Days, AutoClean Error Log Days, AutoClean	These values set a maximum number of days that each category of entry remains in the audit log; older entries are automatically deleted. If the value is 0, the system never deletes entries of that category. Since

Document # SC23-7681-07

Setting	Description
Info Log Days, AutoClean Warning Log Days	string values evaluate to 0 as integers, you can use a value like “Never” instead of 0.
Build Cancel Check Frequency	Specifies how often the system checks for build cancellation requests, in terms of seconds between the checks.
Bump Active Users	When set to N, if an already logged in user (non-root) tries to log in from a new session, Build Forge rejects the login attempt. When set to Y (the default), if an already logged in user (non-root) tries to log in from a new session, Build Forge logs the user out of the previous session, allowing them to log in to the new session. **Note** Regardless of how this setting is defined, a root user who logs in to a new session always bumps another active root user. Only one user can be logged in as user root.
Console URL	Must be set if the console is running on a port other than 80. If set, overrides the default console URL with the value. It takes the form `<protocol>://<hostname>[:<port>]`. Example: `http://myHost:81`.
Create Missing Paths	Default: No. If yes, creates paths for projects if the path is not already present.
Default _AGE	Sets the interval, in seconds, between updates of Run Command properties for server manifests. This value can be overridden by setting a value for _AGE in the server's collector.
Default _MAXJOBS	Sets the default _MAXJOBS value in a collector or manifest. Default: 3.
Default Agent Port	Sets the default port number used for making connections to agents.
Default Import Class	Class to use if an imported project has no defined class or it has a defined class that does not exist. Default: Production.
Expandable Step Log	Determines whether your system uses the standard expanding/contracting view of logs (if you use the default value, Yes) or the new flat log view (if you set it to No) as the default view when displaying step logs.
GDD Secondary Site Failover	Default: No. If Yes, enables secondary site failover for secondary GDD sites.
Hard Run Limit	Default: No. If Yes, the scheduler includes waiting builds in its run limit calculation.
Import Default Secure Access Group	Specifies a default access group for imported projects when the **Import with Secure Access** setting is set to Y.
Import Insecure Default Access Group	Specifies a default access group for imported projects when the **Import with Secure Access** setting is set to N. The default group is used only when the import file lacks an access group.

Document # SC23-7681-07

Setting	Description
Import with Secure Access	When set to Y, the system assigns the default access group listed in the preceding setting to imported data objects. This overrides any access group specified in the XML file you are importing, so that users cannot override security by importing data. When set to N, the system honors any access group settings in imported files.
Inactive server refresh interval	Sets the time, in seconds, between updates of Built-in and Set Value properties for server manifests on inactive servers: servers that are not currently running projects.
Inherit Tag	When set to Yes, causes jobs that are launched via a chain to use the same job tag as their caller. If BUILD_15 of project MasterProject calls project ComponentProject, then the job tag (and the job directory name) for that run of ComponentProject becomes BUILD_15. **Note** The called project always inherits the original tag of the caller; if the caller's tag changes during the run, as a result of a .retag command for example, the called project still gets the tag that the caller started with.
Invalid Relative Dir Characters	Sets the characters that the system will change into underscores if used in project names.
LASTRUN Format	Enter the value for the format for the BF_LASTRUN environment variable, using date format characters as defined for the .date command (see “.date” on page 90).
License Server	The license server hostname. It is set during installation. Example: `myhost.mycompany.com`. The value may include a port number. Example: `myhost.mycompany.com:80`. To change the license server, see "Changing the License Server to Use" in the Installation Guide.
Link Debug Mode	When set to Yes, projects that have source links defined for them run a test of the link instead of running an actual job. The job output for such a run has a single step with output from the adaptor, which you can inspect when troubleshooting your adaptor interfaces.
Link Manual Jobs	Determines whether the system checks source code links when you run a project manually. When set to Y, the system checks the source code link for the project when you launch a job, and may produce change output in the BOM for the job. The system also displays a “Run Link” check box on the Start page for the project, so that you can choose whether to check the source code link when you manually start the job.
Lock Manager Address	Reserved for internal use: IP address and port to use to connect to the lock manager.
Log All Changes	Logs all changes made to all parts of the system. Off is the default. Using this setting has a significant effect on system loading and on database growth. Be sure that you consider its impact when sizing hardware and database capabilities.

Document # SC23-7681-07

Setting	Description
Manifest check interval	Determines how often the system checks for servers whose manifests need to be refreshed, by specifying the number of seconds between checks.
Max Console Procs	Sets the maximum number of processes the console runs at one time. Use as a general throttle on console activity. The system manages processes by storing an ID for each process in the database, and checking the total before launching a new external process. Make sure this value is greater than your **Run Queue Size** setting by at least 5; otherwise the system cannot run enough processes to support the run queue.
Max Inline Depth	Controls the number of levels the system allows for inlining of projects, so that projects cannot be infinitely nested. The default value is 32. If the value is set to 0, the system uses 32. When the system reaches the inline limit, an inlined project which would exceed the limit does not get executed, its steps do not get inserted in the containing project. A message is written to the system messages list: "inline abandoned."
Max simultaneous server tests	Specifies how many server tests can be run at once. Depending on your system resources, running too many server tests at one time can severely slow or lock up the console.
Maximum Refreshes	Maximum number of times that a page refreshes automatically. Default: 50
Override Class when Chaining	Determines whether the system replaces a chained project's class with the class of its caller. A value of Y causes the system to override the chained project's class and use the caller's class instead.
Password Expiration Days	Sets the number of days before users whose passwords are set to expire have to change their passwords. When this time expires, the relevant users are required to change their passwords on next login.
Password Format	Specifies the requirements for user passwords using a format string of six fields separated by periods: *length.req_types.upper.lower.numeric.special* The first two fields specify the following: • Minimum password length (characters) • Minimum number of character types that must be used (an integer ranging from 1 to 4) The remaining fields describe different character type specifications. Each field includes a type and a number. • Type: one of u (uppercase), l (lowercase), n (number), or s (special)

Document # SC23-7681-07

Setting	Description
	• Case: Uppercase (U, L, N, S) indicates that the character is required. Lower case (u, l, n, s) indicates that the character is optional. • Number: for required, indicates the number of characters of this type that are required. For optional, indicates the number of characters of this type that are required *if any are used*. The types are as follows: • U or u to indicate uppercase characters. • L or l to indicate lowercase characters. • N or n to indicate numeric characters. • S or s to indicate special characters. Example: the string 5.2.u1.l1.n1.s1 indicates the following password requirements: • At least 5 characters long • Must include characters from a minimum of two of the four categories (uppercase, lowercase, numeric, special). • For each type, one character of the type qualifies as a match to count toward the requirement. Passwords like abC1x and Abc2% would qualify.
Pause Build Forge Engine	When set to Y, the system completes any current jobs and then pauses the engine. Set it to N to return to normal operation.
Public Hostname	When set, the system substitutes the value of this setting for the server host name in the CONSOLEHOST variable in notification templates
Purge Check Time	Sets the frequency with which the system checks for jobs to purge, in terms of minutes between checks.
QuickReport Public dir	Directory where public report designs are kept. It is a relative path from `<bf-install>/Apache/tomcat/webapps/quickReport`. Default: `../../reports/public`
QuickReport Temp dir	Directory where report designs are created (before being saved elsewhere). It is a relative path from `<bf-install>/Apache/tomcat/webapps/quickReport`. Default: `../../reports/temp`
QuickReport URL	Must be set if the Management Console is running on a port other than 80. It is used by its internals to call Reports > Quick Report, which runs as an Apache Tomcat application (JSP). Use this URL: `http://myHost:8080/quickReport`. Do not use a trailing slash.

Document # SC23-7681-07

Setting	Description
QuickReport User sub directory for custom datasources	User's saved datasources. It is a relative path from `<bf-install>/Apache/tomcat/reports/users/<username>` . Default: `Datasource`
QuickReport User sub directory for saved reports	User's saved report designs. It is a relative path from `<bf-install>/Apache/tomcat/reports/users/<username>` . Default: `reports`
QuickReport User sub directory for uploaded XML files	User's uploaded XML files. It is a relative path from `<bf-install>/Apache/tomcat/reports/users/<username>` . Default: `XML`
QuickReport Users dir	Directory containing all per-user data. It is a relative path from `<bf-install>/Apache/tomcat/webapps/quickReport`. Default: `../../reports/users`
Reload Language Packs	Default: No. If set to Yes, the console reloads its language packs upon restart and resets this value to No. No longer necessary starting in version 7.0.1.
Reset Interface Templates	Use this setting to reset the adaptor templates (to copy changes from an update into your configuration). To use it, set the value to “Yes”, then wait one minute. The system resets the templates and then sets the value back to No.
Reset Server Job-Count	Use this setting (Yes) to simultaneously reset the job count (BF_JOBS) for all servers to zero. The reset occurs when the manifest check interval runs. (The default time is every 10 seconds.) After BF_JOBS has been reset for all servers, the Reset Server Job-Count value reverts back to No (the default).
Run Queue Size	This value limits the number of jobs the system attempts to run at once. When the number of runs in the queue equals or exceeds this number, the system stops moving runs from the Wait queue to the Run queue until the number of jobs drops below this value. If you change your Run Queue Size, check the Max Console Pros setting, which should be greater than the Run Queue Size by at least 5.
Save Start Environ	Controls the default value of the “Save Env” check box on the manual start page for a project. When this setting is Y, the box defaults to checked; otherwise, the box defaults to unchecked. The “Save Env” check box, when checked, causes any changes you make to the environment variables on the Start Page to be saved to the environment records in the database, so that future runs default to those values.
Secondary Site Timeout	If a secondary server fails to respond in this amount of time, the primary console marks it down. The down condition may trigger failover if failover is enabled. Default: 600 seconds.
Server Connect Timeout	Sets the number of seconds the system allows for an attempted connection to a server before giving up.
Server Env Before Chain	Determines whether the calling step's server environment is applied before (Yes) or after (No) it the chained project's project environment is

Document # SC23-7681-07

Setting	Description
	applied. The default value is Yes. Set to No if you want to apply the calling step's server environment after the called project's environment is applied.
Server Retries	Sets how many times the system tries to allocate a step to a server before it gives up and fails the step. If 0, it never gives up.
Server Self-heal time	Sets how long, in seconds, the system waits after an error before trying to use a particular server again.
Server Wait Time	Sets how long, in seconds, the system waits (after it has found no server for a step) before checking again to see if a server is available for the step.
Site-specific selector weight	Weighting for selection purposes in a GDD environment. Default: 10.
SMTP Server	Sets the machine to use as an SMTP server when sending e-mail notifications. The default setting is localhost.
SSO Remote User	Default: No. If Yes, allows single sign-on remote user connections via standard web server authentication.
Stack BuildForge Env Variables	The system normally changes the name of BF_ variables that are passed down to a chained project to BF_CALLER_; this setting determines whether the system *stacks* the naming when chaining goes more than one level deep. The default value is N. When the setting is changed to Y, the BF_TAG variable derived from a calling project two levels deep receives the name BF_CALLER_CALLER_TAG.
Step Log Page Size	Defines how many lines of data to place on each page when the system divides large logs into pages.
Store User Authentication Locally	Determines whether the system caches LDAP/Active Directory user authentication information (in encrypted form). The default value is Yes. The system is only relevant when you use LDAP/Active Directory authentication. When the setting is Yes, the system caches user authentication information in encrypted form, and can use it with the _USE_BFCREDS special variable (which applies user authentication to servers). You may choose to turn off caching by changing the value to No; however, if you do, the system cannot use the _USE_BFCREDS special variable to use the user's credentials when logging into a server.
System Alert Email	The system sends alert e-mail messages to the address defined by this setting. The default is root@localhost.
System Alert Source	When the system sends alert e-mail messages, it uses the address defined in this setting as the sender. The default is root@localhost.

Document # SC23-7681-07

Setting	Description
System Wide Login Message	Allows you to define a message to be displayed above the login form.
System Wide User Message	Allows you to define a message to be displayed at the top of each page, just below the navigation buttons.
Tab Stop	Defines how many spaces the system should use to substitute for tabs when it creates change data reports in BOMs and step logs.
Tag: Date Format	Defines the format used to display the date in the BF_D tag variable. Use the characters y, m, and d as variables for the year, month, and day to show the desired format, along with any desired special characters as separators. For example, for the date September 21, 2005: Format string....Output ymd...050921 m/d/y...09/21/05
Tag: Time Format	Defines the format used to display the date in the BF_T tag variable. Works like the Tag: Date format setting, but uses the characters h, m, and s to stand for hours, minutes, and seconds. The setting h:m:s produces output such as 12:53:42.
Tail Log Amount for Mail Template	Sets the number of lines from the end of a log that are displayed in a notification when the TAILNORMALLOG variable is used in the notification template.
Update Time Vars on Restart	This setting controls whether the system updates system variables when you restart a job.

Messages

The system messages page displays a log of system and user actions that you can review.

Select **Administration → Messages** to display the list of messages. These messages form an audit trail of actions.

You can use the **Severity** box to filter the list by the severity of the messages, and you can limit the list to recent messages by changing the value in the **Last** box.

To display audit messages, select **Audit** in the **Severity** list.

Subscribing to RSS Data Feed for System Messages

The Build Forge product tracks user and system actions and logs these messages to the **Administration → Messages** page.

By default, the Build Forge RSS data feed for system messages displays messages of all severity types and displays the last 15 messages logged.

To subscribe to RSS data feed for system messages, do the following:

Document # SC23-7681-07

1. In the Build Forge Management console, select **Administration** > **Messages** .

   The Web browser detects the RSS data feed and displays an RSS icon in the browser address bar.

2. In the RSS aggregator tool, load the Build Forge RSS data feed.

   For example, copy the URL to add it to the list of RSS data feeds or drag-and-drop the RSS icon to add the URL to the list of RSS data feeds.

3. Subscribe to the RSS data feed to save the URL and be notified when updates occur.

Note

- For details about loading URLs and subscribing to RSS data feeds, consult the documentation for your RSS aggregator tool.
- To view Build Forge system messages or server status through an RSS data feed in languages other than English, your RSS aggregator tool must support UTF-8 multibyte character encoding.

Filtering the RSS Data Feed for System Messages

You can filter the messages displayed in the RSS data feed. To do this, use the filter properties in the **Administration → Messages** page in the Management Console.

For example, change the Severity property to Audit or change the Last property to 12 Hours. When you change a filter property, the URL for the RSS icon on the Messages page is automatically updated.

To load the updated URL into the RSS aggregator tool, right-click the RSS icon on the Messages page to copy the link or drag-and-drop the updated RSS icon to the list of RSS data feeds in the tool.

Note

For details about loading URLs and subscribing to RSS data feeds, consult the documentation for your RSS aggregator tool.

Managing Licenses

When you create a user, the system stores data about the user in the database. After the user logs in to the system, the system assigns a license to the user. You can have two types of users: authorized users and floating users. Authorized users can always log in, but always consume a license. Floating users only consume a license when they are logged in, but may not be able to log in if all the floating licenses are in use.

If you create more users than you have licenses, the new users cannot log in until you purchase additional licenses or use the root user account to remove licenses from some users (or delete some users from the system).

Document # SC23-7681-07

A user can only be logged in once. If you are logged in on one machine, and you log in from another machine under the same name, the original session is invalidated.

Note

If you purchased your system before version 3.8, your license scheme may differ from the one described here. Contact customer support if you have questions about your licensing.

Entering a New License Key

To change your license key, select **Administration → System** and then locate the License Key setting in the list of settings. (Type "license" in the Filter box to quickly display this setting without paging through the list).

Click the License Key item in the list, and the system displays a tab at the bottom of the content pane with an editable **License Key** field. After you edit the field, click the **Save Config** button.

Managing the Engine

This section describes how to pause, start, or stop the engine (`bfengine.exe` / `bfengine.pl`).

Note

If you have any problems getting the engine to run, run it in the foreground so that you can see the status and error messages it generates. On Windows, select **Start → All Programs → IBM Rational Build Forge → Start Engine (Foreground)** to run the engine in the foreground, which causes it to display output in a console window. When the engine is running in the foreground, you can stop it by closing its console window.

Pausing the Engine

If you want to act directly on your Build Forge® database, to back it up, restore it, or change the schema in some way, you should pause the engine before you make any changes.

To pause the engine, select **Administration → System** and locate the **Pause Build Forge Engine** property (use the **Filter** box). The system displays a form for editing the **Pause Build Forge Engine** setting. Change the property to Y and click the **Save Config** button.

The system pauses the engine, but any currently-running projects continue. You must wait for any currently active jobs to complete to ensure that no data is being written to the database.

To reactivate the engine, change the value of the property back to N, and click the **Save Config** button again.

Starting and Stopping the Engine

You may want to stop the engine when you perform a backup of its database or when you install an updated version of the system. The following sections describe how to start and stop the engine on Windows®, Linux® or Solaris.

Document # SC23-7681-07

Starting and Stopping the Engine on Windows

On Windows:

- At **Start → Programs → IBM Rational Build Forge Management Console** , choose either
  - **Start Engine Service**
  - **Stop Engine Service**

Control Panel: You can also use the **Administrative Tools > Services** control panel to start or stop the **IBM Rational Build Forge Management Console** service.

Running in the Foreground: If you have any problems getting the engine to run, run it in the foreground so that you can see the status and error messages it generates: **Start → Programs → IBM Rational Build Forge Management Console → Start Engine (Foreground)** . As the Management Console runs, log output is shown in a console window. To stop the engine in this mode, close its console window.

Starting and Stopping the Engine on Linux or Solaris Systems

If you have an rc file installed (typically in /etc/rc.d/init.d):

```
$ /<path to rc file>/buildforge start
```

```
$ /<path to rc file>/buildforge stop
```

If you do not have an rc file installed, start engine using the following:

```
$ /<path to system installation>/Platform/buildforge.pl &
```

Stop it by determining its process ID and then issuing a kill command:

```
$ ps aux | grep buildforge
$ kill ${<PID>}
```

Managing the Database

This section describes issues that are important when setting up the Management Console database, especially if you want to change default configurations.

Deleting the Database Log File

Delete the database log file regularly.

The system stores database debugging information in a `db.log` file in the Management Console installation directory. You should check the size of this file on a monthly basis, and delete it if you need to free space on the console machine.

Document # SC23-7681-07

Creating a Read-only User for Reporting

Have your database administrator create a user with read-only privileges for use in generating reports from the database.

Error Messages

This section describes error messages you may encounter while using the Management Console.

No active steps

When this message appears in the Next Run column of a schedule entry, all of the steps in the associated project have been disabled. Display the list of steps for the project and enable some of them by clicking on the red circle next to each disabled step.

License key is invalid *or* Build Forge license key is corrupt or missing.

Your license key is either expired or invalid for the product version you have installed. Enter a new license key. See “Entering a New License Key” on page 173.

A database license is required

```
LicMgr: 5140: A database license is required.
```

If the above message appears in your console output (which can be viewed when you run the console in the foreground, or if you view the console log on Linux® or UNIX® systems), you have tried to use an advanced database feature without the benefit of an Enterprise license. For example, you may be trying to use the system with a non-DB2® database. If you want to upgrade your license, contact support.

Document # SC23-7681-07

Integrating with External Applications Using Adaptors

This section describes how to set up and use adaptors with the Management Console. Adaptors are provided for integration with IBM Rational ClearCase and IBM Rational ClearQuest. The Adaptor Toolkit, which allows you to write custom adaptors, is a separately licensed feature.

Adaptor Concepts

This section gives you some general information about adaptors. It also describes how adaptors interact with other Build Forge objects and features.

Get started with adaptors by reviewing the information provided in this section.

Adaptors

An adaptor is an interface to an external application. Adaptors allow a Build Forge project to exchange information with an external application to accomplish a goal.

For example, the adaptor for a source code application checks a repository for source code changes as a prerequisite to running a Build Forge project. If source code files have changed, the project runs. If there are no changes, the project does not run.

Adaptor Templates

An adaptor is an instance of an adaptor template. When you create an adaptor, you assign it a unique name and associate it with a template.

The template is an XML file. The XML contains application commands to gather information, instructions for analyzing information, and format details for displaying results in the BOM report.

The templates provided by Build Forge are designed to be used without modification. But, you can modify templates or use templates as a model for creating a new adaptor template.

The adaptor templates are installed in the following directory:

```
<bf-install>\interface
```

Adaptors and Projects

To execute adaptor code and interface with an external application, the adaptor must be added to a Build Forge project.

Adaptors are added to projects using a dot command or an adaptor link.

Any adaptor can be added to a project using the dot command for the application type: .source, .defect, .test, or .pack.

Document # SC23-7681-07

Only source code adaptors can be added to a project with an adaptor link. An adaptor link is used instead of the .source command to connect the adaptor to the project.

Adaptors and Environment Variables

The adaptor requires environment variables to execute application commands. In the adaptor templates, environment variables are listed in the `<env>` elements in the `<template>` section of the XML file.

For example, for the ClearCaseBaseline adaptor, the following environment variables are listed in the ClearCaseBaseline.xml file:

```
<template>
<!-- Template section, these are parsed out of the final xml.
Use the list below to help identify the variables needed to run this interface
if you are integrating it during a regular BuildForge step.
-->
<env name="VIEW" value="my_adaptor_view" />
<env name="VOB_PATH" value="\AdaptorVob" />
<env name="CCSERVER" value="BFServerName" />
</template>
```

In Build Forge, environment variables are stored in environments. Before creating an adaptor, you create an environment for application environment variables.

Adaptor Links

An adaptor link connects an adaptor to a project and associates an environment to an adaptor.

Adaptor links work only with source code adaptors. You can connect any adaptor type to a project with a dot command.

The adaptor link has the following features:

- Adds adaptor code to the project as step 0 (tests for source code changes *before* running other project steps).
- Automatically populates an environment with application environment variables.
- Allows you to control whether the adaptor code is executed by selecting a state: active, inactive, debug.
- Allows you to run the adaptor as a manually started job.

The adaptor link has the following restriction:

- An adaptor link is defined for one adaptor and one project only. If you wish to use the same adaptor with another project, you must create another instance of the adaptor.

Document # SC23-7681-07

Adaptors and Notification

Most adaptor templates send e-mail notification to users. For example, when the ClearCaseByDate adaptor executes, it sends pass e-mail notification to users who changed source code files. If no files were changed, it sends a fail e-mail notification.

You can optionally modify notification for adaptors:

- In the adaptor template, duplicate the `<adduser>` element to add users to the adaptor notification group.
- In the adaptor template, use the `<notify>` element to add or delete notification messages.
- For adaptor projects, set up project-level notification.
- For adaptor dot-command steps, set up step-level notification.

Adaptors and Job Execution

Adaptor projects that use an adaptor dot-command can be started as a scheduled job or started using any of the manual start options for projects.

Adaptor-linked projects are typically run on a schedule. But, you can manually start an adaptor-linked project if you complete some additional setup. See Manually Starting Adaptor-Linked Projects.

Adaptors and Job Results

For adaptor dot-command projects, view job results in the step log or in the BOM report, as follows:

- Select **Jobs** → **Completed** . Select the tag for the job to view its step log.
- Select **Jobs** → **Completed** . Select the BOM tab to display the BOM report and view job results by category.

For adaptor-linked projects, view job results in the Source Changes category of the BOM report, as follows:

- Select **Jobs** → **Completed** . Select the BOM tab. In the BOM report, locate the Source Changes category.

Adaptor Task Overview

This topic covers all of the tasks required to create a source code adaptor, connect it to a project with an adaptor link, and run the adaptor-linked project in test mode.

Create an Adaptor by Selecting a Template

1. Select **Projects** → **Adaptors.**
2. Click **Add Adaptor** .

Document # SC23-7681-07

3. At Name, enter a unique name for the adaptor. The adaptor name must be unique across the entire set of adaptors.
4. At Type, select the adaptor type.
5. At Template, select the template. The list contains the adaptor templates installed with the Build Forge product.

   ClearCase and ClearQuest adaptors do not require a separate license key. Other adaptors are separately licensed through the Adaptor Toolkit.
6. At Access, select an access group. The ability to view or edit the adaptor is restricted to these group members.
7. Click **Save Adaptor** .

Create an Empty Environment

1. Select **Projects → Environments.**
2. Click **Add Environment**.
3. At Name, enter the environment name. Assign a name that describes the purpose of the environment.
4. At Access, select an access group. The ability to view or edit the environment is restricted to these group members.
5. Click **Save Environment**.

Add the Adaptor to the Project

1. Select **Projects → Adaptor Links.**
2. Click **Add Adaptor Link**.
3. At Adaptor, select the adaptor (and adaptor template) that you created.
4. At Project, select the project. The list displays the projects that are not already linked to an adaptor.
5. At State, select **Active**.
6. At Environment, select the empty environment that you created for the adaptor link.
7. At Populate Environment, select **Yes**. The application environment variables in the adaptor template are added to the environment.
8. Click **Save** to link the adaptor to the project. The adaptor and the project are added to the list of adaptor links.

Document # SC23-7681-07

Edit Environment Variables

1. Select **Environments.**
2. For the environment you created, click the  icon.

   The main content pane displays the adaptor environment variables automatically added to the environment.
3. Review the default values for the environment variables provided by the adaptor template.
4. Change the default values for your source code application as necessary to run the adaptor project.

Run the Adaptor in Test Mode

1. Select **Administration → System.**
2. In the list of system configuration parameters, select **Link Debug Mode** .
3. At Link Debug Mode, select **Yes** .
4. Click **Save**.
5. Select **Jobs → Start.**
6. In the list of projects, select the adaptor-linked project you created from the Start Project page.
7. Click **Execute**.

View Job Status and Logs

To view the job status and log information for the adaptor project:

1. Open **Jobs** .
2. In the list of projects, locate the adaptor-linked project to view job pass/fail status.
3. To view job logs:
   - Select the Tag Name for the adaptor project to access job log information.
   - Select the Bill of Materials to access the BOM report.

Core Adaptor Tasks

This section gives you information about creating and configuring adaptors:

- **Selecting a Template**: describes each of the adaptor application templates.

Document # SC23-7681-07

- **Creating an Environment**: describes options for associating an environment with an adaptor.
- **Creating an Adaptor**: describes how to create an adaptor and associate it with a template.
- **Adding an Adaptor to a Project**: describes options for adding an adaptor to a project.
- **Testing the Adaptor**: describes how to test the adaptor configuration only.

This section also gives you information about other adaptor tasks:

- **Setting the Adaptor Log Level**: describes how to control the quantity of information logged for the adaptor.
- **Quick Start for an Adaptor-Linked Project**: describes setup required for manual start.
- **Resetting Adaptor Templates**: describes when reset is required to update template information.

Selecting an Adaptor Template

Adaptor templates are provided for several application types: source code, defect tracking, testing, and packaging. Adaptors are classified by the type of application they support.

The Build Forge product provides adaptor templates for the applications in following table. The templates for ClearCase and ClearQuest do not require a separate license key. Other application templates are licensed through the Build Forge Adaptor Toolkit.

Adaptor templates are installed in the following directory:

```
<bf-install>/interface
```


Document # SC23-7681-07

Adaptor Template Descriptions

Adaptor Template Name	Description	Type
ClearCaseBaseline.xml	Scans a directory in a ClearCase view. Writes branch and version information reported by ClearCase to the BOM report.	source
ClearCaseByBaselineActivities.xml	Creates a new baseline from the contents of a ClearCase view. Compares the new baseline and the baseline from the previous adaptor execution to identify change activity. For each change activity, writes the following information to the BOM report: activity, files changed, user, date, comments, and version. For each changed file, writes change details (from diff command output) to the BOM report.	source
ClearCaseByBaselineVersions.xml	Creates a new baseline from the contents of a ClearCase view. Compares the new baseline and the baseline from the previous adaptor execution to identify changed files. For each changed file, writes the following information to the BOM report: file name, version, date, user, and comments. For each changed file, writes change details (from diff command output) to the BOM report.	source
ClearCaseByDate.xml	Queries a ClearCase view for changes between two dates. The default dates are the current timestamp and the timestamp of the previous adaptor execution. For each changed file, writes the following information to the BOM report: file name, version, date, user, and comments. For each changed file, writes change details (from diff command output) to the BOM report.	source

Document # SC23-7681-07

Adaptor Template Name	Description	Type
ClearCaseByLabel.xml	Creates and applies a new label to the contents of a ClearCase view. Compares the new label and the label from the previous adaptor execution to identify changed files. For each changed file, writes the following information to the BOM report: file name, version, date, user, and comments. For each changed file, writes change details (from diff command output) to the BOM report.	source
ClearQuestClearCaseByActivity.xml	Finds ClearQuest defect records associated with a list of ClearCase activities. For each defect record found, it adds job information to resolve the defect record within ClearQuest if the ClearQuest status allows it to be resolved. Writes the following information to the BOM report: files associated with ClearCase activity IDs and the ClearQuest defect status.	defect
ClearQuestClearCaseByDate.xml	Queries a ClearCase view for changes between two dates. The default dates are the current timestamp and the timestamp of the previous adaptor execution. For each changed file, looks for a CrmRequest hyperlink attribute that identifies a ClearQuest change ID. Attempts to resolve the change ID by adding job information to resolve the defect record in ClearQuest if the ClearQuest status allows it to be resolved. For each changed file, writes the following information to the BOM report: the file name, defect ID, defect status, and any ClearQuest errors.	defect
CVSv1Baseline.xml	Scans a CVS directory on a Build Forge agent looking for changed files. Writes the following information to the BOM report: changed file name, status, working version, repository version, and sticky tag.	source
CVSv1ByDate.xml	Queries a CVS view for changes between two dates. The default dates are the current timestamp and the timestamp of the previous adaptor execution. Writes the following information to the BOM report: change type, date, user name, version, and file name. For each changed file, writes change details (from diff command output) to the BOM report.	source

Document # SC23-7681-07

Adaptor Template Name	Description	Type
CVSv1ByTag.xml	Applies a new tag to a CVS module. Compares the differences between the newly tagged module and a module tagged during the previous adaptor execution. Writes the following information to the BOM report: file name, revision, state, date, time, change author, and commit comments. For each changed file, writes change details (from diff command output) to the BOM report.	source
CVSv2ByDate.xml	Queries a CVS view for changes between two dates. The default dates are the current timestamp and the timestamp of the previous adaptor execution. Writes the following information to the BOM report: change type, date, user name, version, and file name. For each changed file, writes change details (from diff command output) to the BOM report.	source
PerforceByDate.xml	Queries a Perforce client for changes that occurred since the adaptor execution. Writes the following information to the BOM report: change, date, time, user, Perforce client, and comments. Writes change details (from diff command output) to the BOM report.	source
PerforceByRev.xml	Queries a Perforce client for changes that occurred since the last repository revision. Writes the following information to the BOM report: change, date, time, user, Perforce client, and comments. Writes change details (from diff command output) to the BOM report.	source
Quota.xml	Queries a folder to determine if any of its subfolders exceed a specified threshold size. For each subfolder, writes the following information to the BOM report: folder size, owner, and last modified date. Writes to the BOM report a list of subfolders that exceeded the threshold size.	source

Document # SC23-7681-07

Adaptor Template Name	Description	Type
StarTeamBaseline.xml	Queries the folder for a StarTeam view to gather information about files. Writes the following information to the BOM report: file name, status, revision, and branch.	source
StarTeamByDate.xml	Uses the StarTeam API to query a StarTeam view to identify changes between the current date and the previous adaptor execution. Writes the following information to the BOM report: changed files and directories, user, version, date, and change comments. Writes change details (from diff command output) to the BOM report.	source
SubversionByDate.xml	Queries Subversion for repository changes that occurred between a past date and the current date. Writes the following information to the BOM report: change type, revision, user, file or directory, and change date. Writes the following information to the BOM report: file name, status, revision, and branch.	source
SubversionByRev.xml	Queries Subversion for changes to a repository that occurred between the current revision and an earlier revision. For each change, writes the following information to the BOM report: revision, user, change type, file or directory path, and change date. Writes change details (from diff command output) to the BOM report.	source
VSSByDate.xml	Queries a Visual Source Safe directory for changes between an earlier date and the current date. Writes change information for projects and files to the BOM report: project or file, version, user, date, time, project activity, file project and action. Writes change details (from diff command output) to the BOM report.	source

Creating an Environment for the Adaptor

Adaptors require environment variables to execute application commands. In the adaptor templates, environment variables are listed in the `<env>` element in the `<template>` section of the XML file.

Document # SC23-7681-07

Do not edit environment variables in the adaptor template files. In the Build Forge product, you define environment variables for an adaptor using an environment.

Use an existing environment or create an environment exclusively for adaptor use. Creating one is recommended because it allows you to assign a specific use and descriptive name to the environment; it also simplifies troubleshooting.

This section describes how to associate an environment with an adaptor:

- through an adaptor link.
- in the step associated with an adaptor dot command.

Creating an Environment for an Adaptor Link

Use this method if you are linking the source code adaptor to the project using an adaptor link.

1. Select **Projects** → **Environments** .
2. Click **Add Environment** .
3. A Name, enter the environment name. Assign a name that describes the purpose of the environment.
4. At Access, select an access group. The ability to view or edit the environment is restricted to group members only.
5. Click **Save Environment** .

   Do not add environment variables to the group at this time. They are automatically populated from the adaptor template when you create an adaptor link.

Creating an Environment for Adaptor Dot Commands

Use this method if you are adding the adaptor to a project using an adaptor dot command.

For this task, you need the environment variables for your adaptor.

In the <*adaptor_name*>.xml file, external environment variables are listed in the `<template>/<env>` elements.

Locate adaptor templates in the following directory:

`<bf-install>\interface`

1. Select **Projects** → **Environments** .
2. Click **Add Environment** .
3. At Name, enter the environment name. Assign a name that describes the purpose of the environment.

Document # SC23-7681-07

4. At Access, select an access group. The ability to view or edit the environment is restricted to group members only.
5. Click **Save Environment** .
6. Click **Add Environment Variable** .
7. At Name, enter the environment variable name as it appears in the XML <env> element.
8. At Value, change the substitution variable in the XML <env> element to a real value for your application.

   If you do not know the correct value, you can enter it later.
9. At Action, select **Set** .
10. At On Project, select **Normal** .

Creating an Adaptor from a Template

Every adaptor is a based on an adaptor template. Decide which template you want to use before creating an adaptor.

To create an adaptor, do the following:

1. Select **Projects** → **Adaptors** .
2. Click **Add Adaptor** .
3. At Name, enter a unique name for the adaptor. The adaptor name must be unique across the entire set of adaptors.
4. At Type, select the adaptor type.
5. At Template, select the template. The list contains the adaptor templates installed with the Build Forge product.

   ClearCase and ClearQuest adaptors do not require a separate license key. Other adaptors are separately licensed through the Adaptor Toolkit.
6. At Access, select an access group. The ability to view or edit the adaptor is restricted to these group members.
7. Click **Save Adaptor** .

Adding an Adaptor to a Project

To execute the adaptor code, you must add the adaptor to a project. Create a new project or add the adaptor to an existing project.

This section tells you how to add an adaptor to a project by using the following methods:

Document # SC23-7681-07

- Source code adaptors may be added to a project using an adaptor link.
- Any adaptor (including source code adaptors) can be added to a project using adaptor dot-commands.

Adding a Source Code Adaptor with an Adaptor Link

The Adaptor link connects the adaptor to a project and connects application environment variables to the adaptor.

Before starting this task, create a project and an environment for the adaptor.

After completing this task, open the environment and provide real values for application environment variables if you have not already done so.

1. Select **Projects** → **Adaptor Links** .
2. Click **Add Adaptor Link** .
3. At State, select the state:

State	Description
Active	Runs the adaptor code when the project runs.
Inactive	Skips the adaptor code when the project runs.
Debug	Runs the adaptor code only; skips the other steps when the project runs.

4. At Adaptor, select the adaptor template. The list displays the adaptor templates installed with the Build Forge product.
5. At Project, select the project. The list displays the projects not already linked to an adaptor.
6. Click **Save** to link the adaptor to the project. The adaptor name is added to the list.
7. At Environment, select the environment for the adaptor link.
8. At Populate Environment, select **Yes** . The application environment variables in the adaptor template are added to the environment.
9. Click **Save** to save the adaptor link.

Adding an Adaptor with a Dot Command

Any adaptor can be added to a project using an adaptor dot command. The dot commands calls the <*adaptor_name*>.xml file when the step runs.

Before starting this task, create the project and the environment for the adaptor.

After completing this task, open the environment and provide real values for the application environment variables, if you have not already done so.

Document # SC23-7681-07

To add the adaptor dot command to the project as a step, do the following:

1. Select **Libraries** .
2. In the list, select the project.
3. Click **Add Step**.
4. At Name, enter a step name.
5. At Command, enter the adaptor dot command for the application type: .source, .defect, .test, .pack.
6. At Environment, select the environment created for the adaptor.
7. Click **Save Step**.

Testing the Adaptor

Run the adaptor project to test the adaptor configuration. To verify that the adaptor can interact with the external application and return the expected results, run the adaptor code in isolation from the other project steps.

This section tells you how to test adaptors that are:

- added to a project through an adaptor link.
- added to a project with an adaptor dot-command.

Testing a Linked Adaptor

For source code adaptors linked to the project through an adaptor link, use this procedure to test the adaptor configuration.

The general procedure is as follows:

- make changes to your source files
- run the Build Forge project with the linked adaptor
- check the BOM report for information about changed source files
- check e-mail for pass or fail notification

To test a linked adaptor, do the following:

1. Select **Projects** → **Adaptor Links** .
2. In the list, select the linked adaptor and project.
3. At State, select **Debug** .

Document # SC23-7681-07

4. Click **Save** .
5. In your source code application, make changes to one or more source files. Submit changes to update the source code repository.
6. Run the adaptor-linked project as follows:
    a. Select **Administration** → **System** .
    b. In the list, select **Link Debug Mode** .
    c. At Link Debug Mode, select **Yes** .
    d. Click **Save** .
    e. Select **Jobs** → **Start.**
    f. In the list of projects, select the adaptor-linked project from the Start Project page.
    g. Click **Execute**.
7. Review the BOM report for the job:
    a. Open **Jobs** .
    b. Select the **Completed** tab, then select the **BOM** tab.

Testing an Adaptor Dot Command

For adaptors added to a project with a dot command, use this optional procedure to test the adaptor configuration.

The general procedure is as follows:

- make changes to your source files
- run the Build Forge project with the adaptor dot command
- check the BOM report for information about changed source files
- check e-mail for pass or fail notification

To test the adaptor dot-command, do the following:

1. Select **Jobs** → **Start** .
2. In the list, select the project.
3. Open the **Job Steps** tab.
4. Use the Step Name check box to clear checks for everything except the adaptor dot-command.
5. Click **Execute** to run the project.

Document # SC23-7681-07

6. Review the BOM report for the job:

    a. Select **Jobs** .

    b. Select the **Completed** tab, then select the **BOM** tab.

Setting the Adaptor Log Level

Control how much information is written to the step log for the adaptor by using the _InterfaceLoggingLevel environment variable.

1. Add _InterfaceLoggingLevel to the adaptor's environment.

    - Level 8 logs the most information and level 0 logs the least.
    - Logging levels are inclusive. For example, level 2 includes information from levels 1 and 0.
    - Level 7 is the default logging level.

2. Assign a log level as the value for the _InterfaceLoggingLevel variable:

0: Exec line plus server connection errors or cancel notification; nothing else

1: Parsed commands (commands as they will be sent to the server)

2: Unparsed commands (commands prior to having their local variables set)

3: Build and environment variable SET lines

4: Temp and internal variable SET lines

5: Environment evaluations, e-mail group additions, BOM text logging lines

6: Block and sub-block start/end lines

7: (Default logging level) Agent output that will be checked against match patterns, plus the lines that matched the patterns

8: All agent output

Manually Starting Adaptor-Linked Projects

Adaptor-linked projects can be run on a schedule, run using the quick-start option or started manually if you check the Run Link check box. If you do not check Run Link, the project runs without the adaptor step.

1. Select **Jobs → Start.**

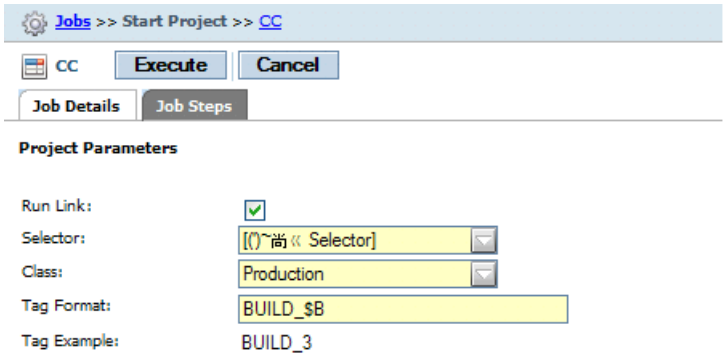
2. In the list of projects, select the name of the adaptor-linked project.

    The Start Project page opens.

Document # SC23-7681-07

3. Check the **Run Link** check box.
4. Click **Execute** from the Start Project page.

Run Link Check Box



Quick Start for Adaptor-Linked Projects

Adaptor-linked projects can be run on a schedule, started manually by selecting the project name from the Start Project page (Jobs > Start), or run using the quick start icon if you complete some additional setup:

1. Select **Administration → System.**
2. In the list, select **Link Manual Jobs.**
3. At Link Manual Jobs, select **Yes.**
4. Click **Save** .
5. Select **Jobs → Start.**
6. In the list of projects, select the Quick Start icon for the adaptor-linked project to run it immediately.

   To verify that the job is running, select the Jobs > Running tab.

Resetting Adaptor Templates

Resetting adaptor templates copies the latest adaptor templates from the `<bf-install>\interface` directory to the Build Forge database.

Reset adaptor templates whenever you:

- install a maintenance release or a new product version
- modify a template by editing the version in the interface directory (not in the Management Console)

Document # SC23-7681-07

- create a new adaptor template

To reset the adaptor templates, do the following:

1. Select **Administration → System** .
2. In the list, select **Reset Interface Templates** .
3. At Reset Interface Templates, select **Yes** .
4. Click **Save** .

Updating ClearQuest Build Records

You can integrate the system with Rational ClearQuest to automatically create and update build records within a ClearQuest database.

The system can automatically create build records in your IBM Rational® ClearQuest® database, with links to the log data. Furthermore, when a job passes, the system can update the ClearQuest database, noting that the job is complete, recording the end time and a summary of the steps that were accomplished. This feature requires Rational ClearQuest version 7.0 or later.

When you configure a project to update a ClearQuest database, the system performs the following:

Job start	When the system launches a job, the system creates a ClearQuest build record. The build record is in the Submitted state and includes the job log URL, the start time, release name, and ID as well as a log entry indicating "Build XYZ started." If a source control adaptor cancels the job (because no source changes are found, for example), no ClearQuest build record is created.
	Note If a project chains another project, the new project gets its own unique ClearQuest build ID.
Job pass/fail	When a job passes or fails, the system changes the build state within ClearQuest to Completed or Failed, sets the build end time, and stores a summary of the job's steps in the ClearQuest build log. The summary includes the name, result status, and server for each step.
Job restart	When a job is restarted, the system changes the build state within ClearQuest to Submitted and creates a ClearQuest build log entry indicating “Build XYZ restarted.”

You configure the ClearQuest integration via special environment variables. To link a project to a ClearQuest database, make sure the following variables are included in the project's environment.

Note

These variables must be present in the project environment; adding them to a step is not sufficient. However, you can use the .include command within an environment variable to

Document # SC23-7681-07

add an environment that includes these variables to another environment, and use that group as the project environment. Also, since the CQ_RELEASE_NAME value is the only one likely to vary per project, you may want to create an environment that contains the other variables, and use .include to include it in the project environment, where you can also specify CQ_RELEASE_NAME as a project-specific environment variable.

Environment Variables Required for ClearQuest Integration

Variable	Description
CQ_DBNAME	Name of the ClearQuest database you want to update.
CQ_INTERACTION	If your project environment has the correct environment variables defined to enable the creation of a ClearQuest build record, but you do not want to create the build record, set this variable to OFF to disable build record creation. To enable build record creation, set this environment variable to ON. **Note** If you are using one of the ClearQuest adaptors, set this environment variable to OFF.
CQ_USER	Username to use when logging into the ClearQuest database
CQ_RELEASE_NAME	The name of the release within the ClearQuest database that you want to update.
CQ_PASSWORD	Password to use when logging into the ClearQuest database. Not required; defaults to blank
CQ_DBSET	The ClearQuest database set value. Not required; defaults to blank.

Advanced Adaptors Tasks

You can modify a Build Forge adaptor template or create a new adaptor template for an external application that you want to interface with a Build Forge project.

Modifying and creating templates are advanced tasks that require a working knowledge of:

- the XML language
- the command language for the external application
- regular expressions

Before working on advanced tasks, read the information in the following sections:

- Core Adaptor Tasks
- Adaptor Template Structure
- Adaptor Reference

Document # SC23-7681-07

The section gives you information about the following tasks:

- **Modifying an Adaptor Template**: describes how to modify a template for all adaptors.
- **Modifying a Template for Single Adaptor**: describes how to modify a template for one adaptor.
- **Creating a New Adaptor Template**: describes the general procedure for creating a new template.
- **Example: Adding a User to Adaptor Notification**: describes how to add an access group to e-mail notification.
- **Example: Removing Change Details from a BOM Report**: describes how to remove change details from the BOM report.

Modifying an Adaptor Template

Use this procedure if you want template modifications to be picked up by all future adaptors created from the adaptor template.

Before you begin, know what you want to modify. For example, you may want to change notifications or modify the BOM report format.

1. Using an XML editor, open the adaptor template that you want to modify.

   Adaptor templates are located in the following directory:

   ```
   <bf-install>\interface
   ```

2. Enter your template modifications.
3. Save the adaptor template. Do **not** change the template name.

The next time you create an adaptor with the modified adaptor template, the new adaptor will include your template modifications.

Modifying a Template for a Single Adaptor

Use this procedure to modify a template for a single instance of an adaptor only.

Before you begin, know what you want to modify. For example, you may want to change notifications or modify the BOM report format.

If the adaptor you want to modify has been created, make your template modifications through Management Console, as follows:

1. Select **Projects** → **Adaptors** .
2. In the text box, enter your modifications to the template.
3. Click **Save Adaptor** .

Document # SC23-7681-07

Note

Your changes are saved only to the instance of the adaptor template associated with the adaptor in the Build Forge database. Changes are not saved to the adaptor template file in the `interface` directory.

If the adaptor you want to modify has not been created, modify the template before you create the adaptor, as follows:

1. Using an XML editor, open the adaptor template.

   Adaptor templates are located in the following directory:

   `<BuildForge_directory>\interface`

2. Enter your modifications to the template.
3. Change the adaptor name and then save the adaptor template.
4. Reset the adaptor templates to pick up your modified template and add it to the list of available templates in the Management Console. See Resetting Adaptor Templates.

Creating a New Adaptor Template

Use this procedure to create a new adaptor template for an external application that you want to interface with a Build Forge project.

1. Review the XML structure and elements in the adaptor templates provided by the Build Forge product.

   Adaptor templates are located in the following directory:

   `<bf-install>\interface`

2. Plan what you want the new adaptor to do:
   - know what commands it will run
   - determine how the commands will be parsed
   - determine what to do with the data gathered from the parsed results
   - know which external environment variables are required
3. Select one of the adaptor templates in the interface directory to use as a model.

   If possible, select an adaptor based on the same external application. Or, select an adaptor that has a function similar to the one you are creating.

Document # SC23-7681-07

Use the XML hierarchy, elements, and element attributes in the model as a guide to create the new template.

4. Using an XML editor, open the model adaptor template. Save the template to the interface directory with a new name.
5. Using your plan, write the XML code for the new adaptor.
6. Save the new adaptor template.
7. Validate the adaptor template using the interface.dtd file in the interface directory.
8. Reset the adaptor templates to pick up your new template and add it to the list of available templates in the Management Console. See Resetting Adaptor Templates.
9. Create a project for the adaptor.
10. Create an environment for the adaptor. See Creating an Environment for the Adaptor.
11. Create an adaptor using the new adaptor template. See Creating an Adaptor from a Template.
12. Add the adaptor to the project. See Adding an Adaptor to a Project.
13. Run the adaptor project to test the adaptor. See Testing the Adaptor.

Creating Multiple Entry Point Adaptors

The adaptor templates provided by Build Forge are single entry point adaptors.

For single entry point adaptors, in the Management Console (Projects > Adaptors), you select the template name for the application and function that you want to execute. For example, ClearCaseBaseline or ClearCaseByDate.

If you prefer you can create one adaptor template for ClearCase that contains multiple interfaces or functions for ClearCase. For a multiple entry point adaptor, you identify each interface by a name, called an entry point.

To create a multiple entry point adaptor:

1. Create the adaptor template.

   To create the template, you have the option of using one of the provided templates as a model and modifying the XML as necessary.

   In the template, you must add the name attribute to the <interface> element to identify the entry point for each interface that you add to the template.

   The related syntax for the interface element is shown in the following example:

```
<interface name="By Date" default="true">
</interface>
```


Document # SC23-7681-07

2. Create an adaptor with a unique name and associate it with the adaptor template. See Creating an Adaptor from a Template.

3. Add the adaptor to the project using an adaptor link or an adaptor dot command.

    - The following example uses the .source adaptor command to add the adaptor to a project step which calls the By Date interface function in the ClearCase adaptor:

      ```
      .source ClearCase "By Date"
      ```

    - To use an adaptor link to call a multiple entry point adaptor, take one of the following actions in the adaptor template to specify which interface function gets executed when the project runs:

        - place the interface element definition for the function to execute as the first interface element in the template file

        - set the default attribute for the interface element to execute to true (default="true")

Example: Enabling E-mail Notification

Adaptor templates can be configured to send e-mail notification to users who cause a change in the external application. The following example shows how to set up two types of notification:

- Notify all users who checked in files for the current build
- Notify all members of a Build Forge access group

The following procedures reference elements in the ClearCaseByDate template. Any adaptor template can use their elements to enable notification.

Notify all users who checked in files for the current build

You can use the `<adduser>` command to dynamically build the group of users who checked in code for the build, then use the `<notify>` command to send notifications to that group.

The ClearCaseByDate template queries ClearCase for a view all changes between two timestamps. The default timestamps are for the current adaptor run and the last adaptor run. In practical terms that translates to a list of all changes since the last build that are checked in for the current build.

Assumption: all user names in the view are known to the SMTP server you use for notification by that name. That means ClearCase user names need to align with e-mail user names.

To enable this notification:

1. Open the ClearCaseByDate adaptor template in an XML editor.

2. Find and edit the `<adduser>` to create a group of users, as follows:

   ```
   <adduser group="MyChangers" user="$4">
   ```

   The positional parameter `$4` refers to the user name field that is shown in the ClearCase view generated by the ClearCaseByDate template.

3. Set up notification to send e-mail to this group. The following setup sends e-mail both when the project fails and when the project succeeds. In some environments you may prefer to notify only if the build fails.

```
<!-- Set some notifications for when the build completes -->
  <onproject result="fail">
   <notify group="MyChangers" subject="Build $BF_TAG ($CurDate) Failed."
message="${Changing}${Changes}"/>
  </onproject>
  <onproject result="pass">
   <notify group="MyChangers" subject="Build $BF_TAG ($CurDate) Passed."
message="${Changing}${Changes}"/>
  </onproject>
```

4. Save the adaptor template.

When the adaptor runs, the MyChangers group is built from the user names in the view. E-mail notification is sent to that group when the build project completes.

Notify all users who belong to a Build Forge access group

In this example you want to notify all members of a Build Forge access group. The ClearCaseByDate adaptor template is used for the example. Assumption: all user names in Build Forge correspond to e-mail user names in the SMTP server.

1. Open the ClearCaseByDate adaptor template in an XML editor.

2. Find and edit the `<adduser>` to create a group of users from a Build Forge access group, as follows:

```
<adduser group="Developer_Access_Group" user="Developer">
```

3. Set up notification to send e-mail to this group. The following setup sends e-mail both when the project fails and when the project succeeds. In some environments you may prefer to notify only if the build fails.

```
<!-- Set some notifications for when the build completes -->
<<<<<<< .mine
  <onproject result="fail">
   <notify group="Developer_Access_Group" subject="Build $BF_TAG ($CurDate) Failed."
 message="${Changing}${Changes}"/>
  </onproject>
  <onproject result="pass">
   <notify group="Developer_Access_Group" subject="Build $BF_TAG ($CurDate) Passed."
 message="${Changing}${Changes}"/>
  </onproject>
```

4. Save the adaptor template.

When the adaptor runs, the Developer_Access_Group group is built from the user names that belong to the Developers access group. E-mail notification is sent to that group when the build project completes.

Document # SC23-7681-07

Example: Removing Change Details from a BOM Report

Most adaptor templates log change details to the BOM report. (The diff command is used to log change details.)

The following steps reference elements in the ClearCaseByDate template, but they can be used to remove change details for any adaptor template.

To remove change details in the BOM report, do the following:

1. Open the adaptor template in an XML editor.
2. Find the `<run>` element that calls the diff command. Remove the following line:

```
<run command="cc_diff" params="$VIEW $1 $2" server="$CCSERVER" dir="/" timeout="360"/>
```

3. Find the `<command>` element for the diff command. Remove the following lines:

```
<!-- The cc_diff command does a generic clearcase diff, logging the full output
of the diff in the BuildForge BOM -->
<execute>
pushd \\view\$1 && cleartool diff -pred -diff_format "$2@@$3"
</execute>
<resultsblock>
<match pattern=".+">
<bom category="Source" section="diff">
<field name="diff" text="$_"/>
</bom>
</match>
<resultsblock/>
</command>
```

4. Find the `<bomformat>` section, then find the `<section>` element for the diff command output. Remove the following lines:

```
<section name="diff">
<field name="diff" title="Change Details"/>
</section>
```

5. Save the adaptor template.

Adaptor Reference

Adaptors are designed to be added to Build Forge projects and run without modification. To modify an adaptor or create a new adaptor, you need to understand the XML template structure and elements used in the Build Forge adaptor templates.

Note

This section does not describe the external application commands used in the Build Forge adaptor templates. For information about these commands, consult the documentation for the external application.

Document # SC23-7681-07

Adaptor templates installed with the Build Forge product are located in the following directory:

```
<bf-install>\interface
```

This section provides the following reference information:

- **Adaptor Requirements**: describes general requirements for using adaptors and specific requirements for ClearQuest adaptor templates.
- **Dot Commands for Adaptors**: describes the syntax for the adaptor dot commands.
- **ClearCase and ClearQuest Environment Variables**: describes the environment variables used by the ClearCase and ClearQuest adaptors.
- **Perforce Environment Variables**: describes some additional environment variables required for Perforce.
- **Adaptor Template Structure**: describes the general structure of the Build Forge adaptor template.
- **Adaptor XML Reference**: describes the XML elements used in the Build Forge adaptor templates.

Adaptor Requirements

This section identifies installation and configuration requirements and software requirements for Build Forge adaptors.

- **General Adaptor Requirements**: requirements for all adaptors.
- **ClearQuest Requirements**: requirements for using the ClearQuestClearCaseByActivity and ClearQuestClearCaseByDate adaptor templates.

General Adaptor Requirements

Verify that your environment meets the requirements for using any adaptor provided with the Build Forge product:

- Install a Build Forge agent on the machine where the external application is running.
- Install a license key for the Build Forge Adaptor Toolkit if you want to use application templates other than ClearCase or ClearQuest.
- Run an external application version that is supported by the Build Forge product.

Document # SC23-7681-07

Application Versions Supported for Adaptors

Application	Version
Rational ClearCase	6.0 or later
Rational ClearQuest	7.0 or later
CVS	1.1, 1.2
Microsoft Visual SourceSafe	6.0
Perforce	2005.1
StarTeam	2005 Release 2
Subversion	1.3.1 and later

ClearQuest Adaptor Requirements

For ClearQuest adaptors, there are additional installation and configuration requirements:

- Install the ClearQuest application on the Management Console system.
- Install the CQperl utility (ClearQuest Perl API) on the Management Console system and add the CQperl installation directory to the system path statement.
- Add the CQ_INTERACTION environment variable to your project and set it to OFF.

Dot Commands for Adaptors

The adaptor dot-commands allow you to add the adaptor for an external application to a Build Forge project as a project step.

The section describes the dot commands for the following application types:

- **.source**: adds the adaptor for a source code application to a project step.
- **.defect**: adds the adaptor for a defect tracking application to a project step.
- **.test**: adds the adaptor for a testing application to a project step.
- **.pack**: adds the adaptor for a packaging application to a project step.

.source

Description Use the .source command to add an adaptor for a source code application to a project step. A source code adaptor is a Build Forge object; it is based on the adaptor template for a source code application. The adaptor code for the step is executed when the project runs.

Syntax `.source <adaptor_name> [entry_name]`

Document # SC23-7681-07

The <*adaptor_name*> is required; it is the name assigned to the adaptor in the Management Console. The <*adaptor_name*> case should match the case used in the console.
If your adaptor template has multiple interface functions, specify the one that you want to execute by using the *entry_name* option. The *entry_name* must match the name attribute specified for interface element in your adaptor template. In the following example, the entry name is By Date.
If you are using an adaptor link, the adaptor is called automatically and the first interface function in the adaptor template is executed. To execute a different interface, edit the adaptor template to set the default attribute to true (default="true") on the interface that you want to execute.

Examples

```
.source MyClearCaseAdaptor
```

```
.source MyClearCaseAdaptor "By Date"
```

Notes To create an adaptor or view a list of adaptors, select **Projects → Adaptors** .
The adaptor templates provided with the Build Forge product are located in:

<*bf-install*>/interface

.defect

Description Use the .defect command to add an adaptor for a defect tracking application to a project step. A defect adaptor is a Build Forge object; it is based on the adaptor template for a defect tracking application. The adaptor code for the step is executed when the project runs.

Syntax `.defect <adaptor_name> [entry_name]`

The <*adaptor_name*> is required; it is the name assigned to the adaptor in the Management Console. The <*adaptor_name*> case should match the case used in the console.
If your adaptor template has multiple interface functions, specify the one that you want to execute by using the *entry_name* option. The *entry_name* must match the name attribute specified for interface element in your adaptor template. In the following example, the entry name is DefectFunction.

Examples

```
.defect MyClearCaseQuestAdaptor
```

```
.defect MyClearCaseQuestAdaptor DefectFunction
```

Notes To create an adaptor or view a list of adaptors, select **Projects → Adaptors** .
The adaptor templates provided with the Build Forge product are located in:

<*bf-install*>/interface

.test

Description Use the .test command to add an adaptor for a test application to a project step. A test adaptor is a Build Forge object; it is based on the adaptor template for a test application. The adaptor code for the step is executed when the project runs.

Syntax `.test <adaptor_name> [entry_name]`

The <*adaptor_name*> is required; it is the name assigned to the adaptor in the Management Console. The <*adaptor_name*> case should match the case used in the console.
If your adaptor template has multiple interface functions, specify the one that you want to execute by using the *entry_name* option. The *entry_name* must match the name attribute specified for interface element in your adaptor template. In the following example, the entry name is TestFunction.
To execute a different interface, edit the adaptor template to set the default attribute to true (default="true") on the interface that you want to execute.

Examples
```
.test MyTestAdaptor
```
```
.test MyTestAdaptor TestFunction
```

Notes To create an adaptor or view a list of adaptors, select **Projects** → **Adaptors** .
The adaptor templates provided with the Build Forge product are located in:

`<bf-install>/interface`

.pack

Description Use the .pack command to add an adaptor for a packaging application to a project step. A packaging adaptor is a Build Forge object; it is based on the adaptor template for a packaging application. The adaptor code for the step is executed when the project runs.

Syntax `.pack <adaptor_name> [entry_name]`

The <*adaptor_name*> is required; it is the name assigned to the adaptor in the Management Console. The <*adaptor_name*> case should match the case used in the console.
If your adaptor template has multiple interface functions, specify the one that you want to execute by using the *entry_name* option. The *entry_name* must match the name attribute specified for interface element in your adaptor template. In the following example, the entry name is PackageFunction.
To execute a different interface, edit the adaptor template to set the default attribute to true (default="true") on the interface that you want to execute.

Examples
```
.pack MyPackagingAdaptor
```
```
.pack MyPackagingAdaptor PackageFunction
```


Document # SC23-7681-07

Notes To create an adaptor or view a list of adaptors, select **Projects** → **Adaptors** .
The adaptor templates provided with the Build Forge product are located in:

`<bf-install>/interface`

ClearCase and ClearQuest Environment Variables

The ClearCase and ClearQuest adaptor templates use the environment variables in the following table to execute cleartool commands.

Not every environment variable in the following table is required for each ClearCase or ClearQuest adaptor template.

In each adaptor template, required environment variables are listed in the `<env>` elements in the `<template>` section.

Before running a ClearCase or ClearQuest adaptor project, supply real values for the required variables or accept defaults. Edit variable values in the environment assigned to the adaptor.

Adaptor templates are located in the following directory:

`<bf-install>\interface`

Document # SC23-7681-07

Environment Variables for ClearCase and ClearQuest

Environment Variable Name	Substitution Variable	Description
VIEW	value=my_adaptor_view	Set this variable to the name of the ClearCase view that you want to use with the adaptor.
VOB_PATH	value=\c_vob	Set this value to the name of your component VOB, and optionally, its subdirectories.
PROJECT_VOB	value=\ProjectVob	When you use the ByBaseline adaptor, set this variable to the name of your Project VOB (only used with UCM ClearCase).
CCSERVER	value=BFServerName	Set this variable to the name of a Build Forge server that has the ClearCase client installed and running.
CurDate	value=.date %d-%b-%y.%H:%M:%S	Supplies the current date to the adaptor, using a .date command to generate the date in the format expected by ClearCase. Do not change this value.
LAST_RUN	value=1-Jan-05.00:00:00	For ByDate adaptors, the system uses this value to determine whether any changes have occurred; it's the date of the last successful run. You can manipulate this value when testing the adaptor to force the adaptor to run, by picking a date that you know precedes some changes. If the adaptor allows run to continue, it automatically updates this value to the current date. The default value is 1-Jan-05.00:00:00.
LABEL	value=BUILD_1	For ByLabel adaptors, when you use your adaptor to generate differences by label (with the ByLabel adaptor), the system uses this value as the label.
BASELINE	value=BUILD_1	For ByBaseline adaptors, when you use your adaptor to generate differences by baseline, the system uses this value as the baseline.
ACTIVITIES	value=SAMPL00000001@\ProjectVob	For the ClearQuestClearCaseByActivity adaptor, a space-delimited set of activity IDs.

Perforce Environment Variables

Add the following Perforce environment variables to the Build Forge environment group that is assigned to the Perforce adaptor:

- P4USER
- P4PASSWD

To access the Perforce server, Build Forge requires a valid user name and password. In the step log, the Perforce user name and password are written in plain text.

Document # SC23-7681-07

The Assign Hidden property for environment variables cannot be used to encrypt Perforce authentication information.

Adaptor Template Structure

This topic describes the general XML structure or element hierarchy in the Build Forge adaptor templates.

The adaptor template is made up of the following section elements: `<template>`, `<interface>`, `<command>`, and `<bomformat>`. Each of the these sections contains child elements.

For element descriptions, see the Adaptor XML Reference.

```
<PROJECT_INTERFACE>
<template>
<env/>
</template>

<interface>
<setenv/>
<run>
<ontempenv>
<step/>
</ontempenv>
<onproject>
<notify/>
</onproject>
</interface>

<command>
<execute> or <command>
command line
</execute> or </command>
<resultsblock>
<match>
<bom>
<field/>
</bom>
<adduser/>
<setenv/>
<run/>
</match>
</resultsblock>
</command>

<bomformat>
<section>
<field/>
</section>
</bomformat>
</PROJECT_INTERFACE>
```

Adaptor XML Reference

This section describes the elements used in the adaptor XML language. Elements are listed in approximately the order they appear in an adaptor XML file. Some examples and pseudocode are included in the descriptions.

Document # SC23-7681-07

PROJECT_INTERFACE

The <PROJECT_INTERFACE> element wraps all the other tags in the adaptor template. It takes one attribute, IFTYPE, which indicates the adaptor type; valid types include Source, Test, and Defect.

```
<PROJECT_INTERFACE IFTYPE="Source">
...all other elements...
</PROJECT_INTERFACE>
```

env

The <env> element is used inside a <template> element to define environment variables (with initial values) that can be copied into the environment used with a project link. Each <env> element should include *name* and *value* properties. The value provides an initial value for the variable.

```
<env name="FILESPEC" value="//depot..." />
```

setenv

Use the <setenv> element to initialize environment variable values within <interface> or <match> elements. The <setenv> element is an empty element; it does not contain other elements. The following example shows the attributes for a <setenv> element:

```
<setenv group="Adaptor" name="LAST_VER"
   value="$LAST_VER>$1?$LAST_VER:$1" eval="true"/>
```

The optional eval attribute identifies whether the value should be logically evaluated; in the example, the LAST_VER variable will be set to the greater of $LAST_VER or the value in the $1 variable. An additional attribute, type, can be used if you do not use the group attribute.

You can use a variable in the value of the name attribute, so that you can set or create variables whose names are not known in advance.

The element can be used in three different ways:

- When you specify a group name, this element works like the .set command: it sets the variable value in the master record in the database, not the copy used by the current step, so the change is not seen by the adaptor algorithm, but takes effect in the first step of the next job (or the next time that the group is copied from the master record, which can happen if a step explicitly references the group). When you specify a group name, you must refer to an existing variable within the specified group; you cannot create a new variable.

  <setenv group="MyGroup" name="LAST_RUN" value="$CurDate"/>

  ### Note

  If you are creating or editing an adaptor template, you may use the [ADAPTOR] placeholder instead of specifying a static environment for the <setenv> group attribute.

  ```
  <setenv group="[ADAPTOR]" name="MyVar" value="MyVal"
  ```

  When the project runs, Build Forge looks for the environment associated with the adaptor link or the adaptor dot command and uses this environment as the value for the group="[ADAPTOR]" attribute.

Document # SC23-7681-07

- When you do not specify a group name, the element works like the .bset command: it sets the variable value in the job's copy of the project environment. The change is available to all the steps in the job, but does not affect the master record for that environment: a later job would not see the change. You can create new variables with this form.

  ```
  <setenv name="NewDriverNeeded" value="Y"/>
  ```

- When you skip the group name and specify a temporary variable (type="temp"), the element sets up a temporary variable for the use of the adaptor logic only; the variable does not persist past the special adaptor step. You can create new variables with this form.

  ```
  <setenv name="Changing" value="Source Changes for this Job:\n" type="temp"/>
  ```

  You have two other options with temporary variables:

  - You can specify the type as “temp append”, which causes the system to append the value to the existing value. Any characters you add after “append” are placed between the new and old values; the following example uses \n to insert a newline before adding some data:

    ```
    <setenv name="Changes" value="$4 - $1 - $6" type="temp append\n" />
    ```

  - You can specify the type as “temp once” to create a temporary variable that can only be set once. After the variable is set the first time, the system does not allow it to be reset during that job of the adaptor.

    ```
    <setenv name="BaseFolder" value="$1" type="temp once"/>
    ```

Use the conditional attribute to control whether the `<setenv>` element initializes or modifies environment variables in `<interface>` or `<match>` elements. The value of the conditional attribute is an expression that evaluates to true or false. If the expression evaluates to true, `<setenv>` initializes or modifies environment variables; otherwise, it does not.

run

A `<run>` element is used within an `<interface>` element to specify a named command to run. The command is defined later in the same XML file. The `<run>` element is an empty element. A run element must contain the following attributes:

- A params attribute to specify the parameters to pass to the command, as a string.
- A server attribute to specify a server to run the command on.
- A dir attribute to specify the path (under the server's path) in which to run the command.

```
<run command="p4_changes" params="$LAST_RUN $FILESPEC $LAST_VER"
server="$P4CLIENT" dir="/" timeout="360"/>

<run command="UpdateEnv" params="" server="" dir="/" timeout="360"/>
```

Use the mode attribute to identified the `<run>` mode for the named `<run>` command. The mode attribute takes the following values:

Document # SC23-7681-07

- **conjoined**: all calls to the command are grouped in one call for server processing.
- **parallel**: calls are processed individually as server slots become available.
- **exec**: commands are started and immediately processed by the server.

Use the conditional attribute to control whether the <run> command is initialized. The value of the conditional attribute is an expression that evaluates to true or false. If the expression evaluates to true, the command is initialized; otherwise, it is not executed.

ontempenv

The <ontempenv> element is used within an <interface> element and acts like an if-then statement. Use this element to return a pass or fail value to the project; a pass indicates that the system should continue and run the rest of the project, while a fail indicates it should stop. This is normally used to indicate whether the interface found relevant changes that merit a new run of the project.

After the system executes any commands specified in <run> elements, it processes the <ontempenv> element. Use the name attribute of this element to specify a temporary environment variable, and the state attribute to specify a value.

The <ontempenv> wraps a <step> element, which is executed only if the temporary environment variable name and state exists after the <run> element commands are executed.

```
<ontempenv name="Changes" state="empty">
  <step result="FAIL"/>
</ontempenv>
```

step

The <step> element is used only within the <ontempenv> element. It specifies the outcome of the special adaptor step. It is an empty element. The follow examples show the two forms of the <step> element.

```
<step result="FAIL"/>
```

```
<step result="pass"/>
```

onproject

The <onproject> element defines notification actions that are performed by the system after the system executes the project steps. The element takes a required result attribute which specifies whether the actions are performed for a passing or failing job. Typically, an adaptor XML file contains two <onproject> elements, one for the pass case and one for the failure case. The following example shows a pair of <onproject> elements that use notify elements to send different messages depending on whether the project passes or fails:

```
<onproject result="fail">
      <notify group="MyChangers"
      subject="Run $BF_TAG ($CurDate) Failed." message="$Changing$Changes"/>
</onproject>

<onproject result="pass">
     <notify group="MyChangers"
```


Document # SC23-7681-07

```
     subject="Run $BF_TAG ($CurDate) Passed." message="$Changing$Changes"/>
</onproject>
```

command

An adaptor XML file can contain several `<command>` elements; each defines a named command that can be referenced by `<run>` elements within `<interface>` elements. The `<command>` elements are specified outside the <interface> elements so that multiple interfaces within an XML file can reuse the same commands.

Commands can call other commands by embedding a `<run>` command in the `match` element in the `<resultsblock>` elements.

The <command> element wraps a structure of `<execute>` and `<resultsblock>` elements as shown below:

```
<command name="p4_changes">
   <execute>
       command line
   </execute>
   <resultsblock>
      Has its own structure.
   </resultsblock>
</command>
```

Alternatively, the `<execute>` element can be replaced in the block with an `<integrate>` element.

Use the mode attribute to identified the mode for the `command` element. The mode attribute takes the following values:

- **conjoined**: all calls to the command are grouped in one call for server processing.
- **parallel**: calls are processed individually as server slots become available.
- **exec**: commands are started and immediately processed by the server.

execute

The <execute> element is used inside the <command> element to specify commands. The contents of the element are one or more lines of text to be sent to the server used by the adaptor. Dot commands cannot be used in the <execute> element. When a <run> element calls a <command> element, the system replaces any positional parameters in the <execute> element's content with the parameters specified in the calling <run> element. A $1 parameter in the <execute> element's content is replaced by the first parameter, a $2 parameter is replaced with the second parameter, and so on.

Use the conditional attribute to control the execution of the commands in the <execute> element. The value of the conditional attribute is an expression that evaluates to true or false. If the expression evaluates to true, the system executes the commands; if the express evaluates to false, the commands are not executed.

```
<execute>
p4 changes -s submitted -t -i $2@$1,@now
</execute>
```


Document # SC23-7681-07

integrate

The <integrate> element is similar to the <execute> element and can be used in its place. Like the <execute> element, the <integrate> element specifies a command line to be executed. It has the following differences:

- The command line is executed on the Management Console system, not the server that executes the adaptor.
- The command line uses the `\integration` directory (a subdirectory of the installation directory) as its current directory.

The <integrate> element is useful for executing applications or scripts located on the Management Console machine, especially in the `\integration` directory.

When a <run> element calls the <command> element that contains the <integrate> element, the system replaces any positional variables in the <integrate> element with the parameters specified in the calling <run> element. A $1 in the <integrate> element is replaced by the first parameter, a $2 with the second parameter, and so on.

As with the <execute> element, you cannot use dot commands in an <integrate> element.

The following example, from the IBM Rational ClearQuest adaptor, sends data to ClearQuest by running the CQperl command (a ClearQuest program for executing Perl code) and feeding it the name of a Perl script located in the`\integration` directory. The example assumes ClearQuest is installed on the Management Console system.

```
<integrate>
cqperl bfcqresolve.pl $2 Fixed "Fixed in build $BF_TAG"
</integrate>
```

resultsblock

The `<resultsblock>` element defines how the system should process the results of the command lines executed from the related `<execute>` element. The `<resultsblock>` element is only used within a <command> element. The `<resultblock>` element can be nested to partition results.

The `<resultsblock>` element can have optional beginpattern and endpattern attributes that use Perl regular expressions to define a range of output lines to process. You can use this to have some portions of the output processed by different `<resultsblock>` elements. The following pseudocode shows the structure of a `<resultsblock>`.

```
<resultsblock startpattern="" endpattern="" >
        <match>
        <bom>
         <field/>
        </bom>
        <adduser/>
        <setenv/>
        <run/> (The <run> element can be used to run commands within other commands)
        </match>
        <setenv/>
      </resultsblock>
```


Document # SC23-7681-07

The following example shows how the <resultsblock>, <match>, and <bom> elements work together:

```
<resultsblock
beginpattern="^Change (\d+) by (.*?)@(.*?) on (.*?) (.*?)$"
endpattern="^Differences ...$">
        <match pattern="(?=^(?:(?!Differences ...).)*$).?">
        <bom category="Detail" section="descriptions">
        <field name="Description" text="$_"/>
        </bom>
        </match>
     </resultsblock>
```

match

A <match> element is used within a <resultsblock> element to process lines of output. The <match> element takes a pattern attribute that defines matching lines. The pattern is a Perl regular expression.

The match pattern can include parenthetical expressions, which are stored in the variables $1...$n.

```
<match pattern="^Change (\d+) on (.*?) (.*?) by (.*?)@(.*?) '(.*?)'$">
```

The <match> element uses <bom> and <field> as subelements. See “resultsblock” on page 212 for a more extensive example.

bom

A <bom> element defines information to be logged to the Bill of Materials (BOM) for the job; it should be enclosed in a <match> element. A <bom> element must specify a category and section within the BOM, and defines which numbered variables ($1...$n) collected by the <match> element should be converted to fields for the BOM data.

```
<bom category="Source" section="changesets" >
        <field name="Change" text="$1"/>
        <field name="Date" text="$2"/>
        <field name="User" text="$4"/>
</bom>
```

Use the conditional attribute to control whether the <bom> element is written to the BOM report. The value of the conditional attribute is an expression that evaluates to true or false. If the expression evaluates to true, the information in the bom element is written to the BOM; otherwise, it is not written.

field

The <field> element can be used within either the <bom> or <section> elements to specify a field.

When used in a <bom> element, specify the name and text; the text attribute defines which variable is used to populate the field with data.

When used in a <bomformat> <section> element, specify the name and title. The name specifies the logical name, while the title is used for display. If there is more than one field in a <section>, include an order attribute.

```
<section name="changesets">
        <field order="1" name="Change" title="Change ID"/>
        <field order="2" name="Date" title="Date"/>
        <field order="3" name="Time" title="Time"/>
        <field order="4" name="User" title="User ID"/>
        <field order="5" name="Client" title="Client"/>
        <field order="6" name="Comment" title="Comment"/>
</section>
```

Use the conditional attribute to control whether the `<field>` element is written to the BOM report. The value of the conditional attribute is an expression that evaluates to true or false. If the expression evaluates to true, the information in the `<field>` element is written to the BOM; otherwise, it is not excluded.

Use the template attribute to define the text format for the `field` element. For example, if the text is a string, the template value might be "Hello $VALUE". When the field is written to the BOM report, $VALUE is replaced with the field text.

adduser

The `<adduser>` element is used within a `<match>` element to add users to a temporary group based on the output of change commands, so that the adaptor can send notifications to users who caused changes. The system does not add a user to a group if the user is already a member of the group, preventing multiple notifications. The `<adduser>` element is an empty element. The group attribute specifies a temporary group created during the adaptor logic run; you must reference the same group in a `<notify>` element to cause the actual notifications to be sent.

```
<adduser group="MyChangers" user="$4"/>
```

Use the conditional attribute to control whether the `<adduser>` element adds users who caused changes to a temporary access group. The value of the conditional attribute is an expression that evaluates to true or false. If the expression evaluates to true, the temporary access group is created; otherwise, it is not created.

bomformat

The `<bomformat>` element defines how to display the data collected in earlier `<bom>` elements. It takes a category attribute that specifies the logical name of a BOM category as well as title attribute to specify the display name for the category. A structure of `<section>` elements which contain `<field>` elements defines the layout, as in the following example:

```
<bomformat category="Detail" title="Change Details">
        <section name="descriptions" parent="section name" expandable="yes">
        <field name="Description" title="Change Description"/>
        </section>
        <section name="diff">
        <field name="Diff" title="Differences"/>
        </section>
```

section

The `<section>` element defines how to display a portion of a BOM category. It takes a name attribute. The `<section>` element can only be used within `<bomformat>` elements.

Document # SC23-7681-07

Accessing Management Console from IDEs with Plug-ins

This section describes how to download and set up API packages that allow you to access Management Console features from an external application. The API toolkit is a separately licensed feature.

About Plug-ins for IDEs

You can use plug-ins to connect to a Management Console from your integrated development environment (IDE).

The plug-ins allow developers to access Management Console features from a supported integrated development environment (IDE) on a client machine (a developer's personal machine, rather than the Management Console.

The set of projects that a developer can access through the plug-ins is defined by the access groups that the developer user is a member of. If the steps within a project have different access groups assigned, that can also limit the developer's access. Note that the plug-ins do not allow developer users to change projects or steps, no matter what their access properties, so you can freely place developers in access groups to allow them to launch or view projects.

Each user who wants to use a plug-in needs a license.

This release of the plug-ins supports the Eclipse™, Rational® Application Developer, and Microsoft Visual Studio IDEs.

Plug-ins for Eclipse and Rational Application Developer

Plug-ins provide access to Management Console features from within Eclipse™ or Rational® Application Developer IDEs.

The following plug-ins are available for the Eclipse and Rational Application Developer environments:

Available Plug-ins

Frequency

The Frequency plug-in provides developer self-service features. Through it, a developer can launch and view those projects allowed by his access groups.

When it is installed in your IDE, you can perform the following actions:

- Access one or more Management Consoles to view preconfigured projects
- View build logs

Document # SC23-7681-07

- Launch builds (including test builds using the developer's local files)
- View status of running builds

Reflector — The Reflector plug-in allows developers to run test builds: builds that use files from the developer's local workspace in a remote build configured through the management console. This allows you to test code changes against the actual production build process without checking them in, or see how your local code performs in a test, integration, or release build, without impacting others. You can even apply environment variable overrides, so that different values are used when you run your version of the project.

Eclipse plug-in users have the option to override values for project environment variables. The Job Settings pop-up displays when you start a Build Forge project. Changes to environment variables apply to the job only; default variables values for the project are not changed.

Using the Plug-ins in Eclipse or Rational Application Developer

After you install the plug-ins, you can activate them in the following ways:

- To access Management Consoles to launch jobs and view project logs, use the Frequency plug-in. *Within your IDE*, select **Window → Open Perspective → Other** . Your IDE displays a dialog box with a list of perspective types; select the Build Forge perspective. The system displays the Console Explorer, Build Info, and Build Log windows. Right-click on the **Console Explorer** and select **New Console** to configure a connection to a Management Console. For more information on using Frequency, see the on-line help provided with the plug-in.

  ### Note

  If you need to configure access to an LDAP/Active Directory domain, make sure you use the Build Forge system name for the LDAP domain object, not the actual name of the domain.

- To run test builds, use the Reflector plug-in. *Within your IDE*, configure Reflector by selecting your project and right-clicking. Select **Properties** from the pop-up menu. In the **Properties** dialog's list of properties options, select **Build Forge Project Artifacts** . Configure the dialog with the Build Forge project you want your project to work with, and select files to be uploaded to the system. For more information, see the on-line help provided with the plug-in.

  ### Note

  The Reflector plug-in requires the Frequency plug-in.

Installing the Eclipse or Rational Application Developer Plug-ins

Install the plug-ins from your IDE environment by connecting to an update site.

Use the following procedure to install the plug-ins in Eclipse™ 3.02 or later, or Rational® Application Developer version 6.0 or later. Perform these steps from within your IDE.

Document # SC23-7681-07

Note

The Build Forge system must be running.

Note

The Eclipse plug-ins require Eclipse version 3.02 or later. Eclipse is a free, open-source IDE. You can download Eclipse from http://www.eclipse.org.

1. Select **Help** → **Software Updates** → **Find and Install** .
2. Click the **Search for new features to install** radio button.

   The system displays the **Update Sites to Visit** dialog.
3. Click the **New Remote Site** button.

   The system displays the **New Remote Site** dialog.

   a. Enter “Build Forge Plug-In Update Site” in the name field.

   b. Enter the following update site URL in the **URL** field, using the hostname of your Management Console machine: `http://<console_host_name>/prism/site.xml`

   c. Click **OK** .
4. The system displays a list of available plug-ins. Select all the offered plug-ins.

   ### Note

   The Reflector plug-in requires the Frequency plug-in.
5. In the **Update Sites to Visit** dialog, select the new **Build Forge** check box and click its + sign to expand it. Verify that the **Build Forge Prism Plug-ins** check box is selected.
6. Click **Next** .
7. Respond to the prompts provided by the installation script.

Plug-in for Microsoft Visual Studio

The plug-in provides access to Management Console features from within Microsoft Visual Studio IDEs. This release supports Visual Studio 2003 and Visual Studio 2005.

You can access the following features via the Visual Studio plug-in:

- Access one or more Management Consoles to view preconfigured projects (as provided by the security settings relevant to your Build Forge login).
- View build logs.
- Launch builds.

- View status of running builds.
- Run *pre-flight* builds: builds that use files from the developer's local workspace in a remote build configured through the management console. This allows you to test code changes against the actual production build process without checking them in, or see how your local code performs in a test, integration, or release build, without impacting others.

Using the Plug-in

After you install the plug-in, start Microsoft Visual Studio. The **Console Explorer** , **Build Info** , and **Build Log** windows appear.

- To access Management Consoles to launch jobs and view project logs, use the plug-in's windows inside Visual Studio. Configure access to a console in the **Console Explorer** ; and view jobs and logs in the **Build Info** and **Build Log** windows. For more information on using Frequency, see the help provided with the plug-in.
- To run pre-flight builds, *within your IDE* configure your project by selecting your project and right-clicking. Select **BuildForge Project Settings** from the pop-up menu. For more information, see the help provided with the plug-in.

Installing the .NET IDE Plug-in

Install the plug-in by running its .msi file and then starting your IDE.

Use the following procedure to install the plug-in on a client system with Microsoft Visual Studio 2003 or 2005 installed.

1. If you have an earlier version of the plug-in installed, uninstall it (using **Start → Control Panel → Add/Remove Programs**).
2. Exit Visual Studio.
3. Download a copy of the `BuildForgeVSNETSetup.msi` file. Double-click it to start the installation.

   If you are installing for Visual Studio 2003, you can choose the installation location. For Visual Studio 2005, the plug-in is automatically installed to the required location.
4. Start Visual Studio.

   Your Visual Studio window displays the **Console Explorer** , **Build Info** , and **Build Log** windows which derive from the plug-in. The plug-in appears as a line item on the Add-in Manager tool (**Tools → Add-in Manager**) and on the options panel (**Tools → Options → BuildForge**). By default, the plug-in's windows are displayed whenever you start up Visual Studio.

   ## Note

   If you close the plug-in's windows, you can redisplay them by selecting **Tools → Options → BuildForge** and making sure the check box for each window is selected, then clicking **OK** .

Special Variables for Test Projects

When you run a test build of a project using a plugin, you can use some special environment variables to run commands before and after files from your system are copied to the server.

All commands are run in the project directory:

- PRECMD variables are used to run a command on directories and files that are copied from the developer's machine to the server running the build. The command runs before the project step. Example: this command could check out files from a source control system before they are copied.
- POSTCMD variables are used to run a command on directories and files after a project step has executed. Example: this command could be used to free a checked-out virtual directory (in a source control system that uses such a concept, like Rational ClearCase).

Commands are run on directories and files marked in a Reflector plug-in as Build Forge Project Artifacts. The commands are applied as the directory tree for the reflector is traversed.

Note

Traversal of the directory tree is breadth-first downward for PRECMD commands and reversed for POSTCMD commands. Commands for directories and commands for files are run as appropriate during traversal.

_PRISM_DIR_PRECMD	Specifies a command to be run on directories as they are encountered during tree traversal. The command is run once for every directory that contains at least one file. If the `$1` token is used in the command, it is replaced the name of the directory the command is running on. Only the first occurrence of `$1` is handled this way.
_PRISM_FILE_PRECMD	Specifies a command to be run on files as they are encountered during tree traversal. The command is run once for every file. If the `$1` token is used in the command, it is replaced the name of the file the command is running on. Only the first occurrence of `$1` is handled this way.
_PRISM_DIR_POSTCMD	Specifies a command to be run on directories as they are encountered during tree traversal. is run once for every directory that contains at least one file. The command is run once for every directory in the project that contains at least one file. If the `$1` token is used in the command, it is replaced the name of the directory the command is running on. Only the first occurrence of `$1` is handled this way.
_PRISM_FILE_POSTCMD	Specifies a command to be run on files as they are encountered during tree traversal. The command is run once for every file. The system replaces the first `$1` in the command with the file name.

Document # SC23-7681-07

Working with APIs

This section describes how to download and set up API packages that allow you to access Management Console features from an external application.

APIs

Build Forge provides a Java client API and a Perl client API.

The Build Forge APIs are included with the Enterprise Edition.

Two client interfaces are provided:

- Java Client
- Perl Client

Client files are stored in `<bf-install>/webroot/public/clients/.`

You can access the files on a running Management Console. The Client download directory is at the following URL: `http://<hostname>:<portnumber>/clients/`

Creating a Build Forge User for API Programs

Create a user on the Management Console for programs to use to authenticate.

Create a user for API programs to use for logging into the Management Console. Log in to test the user to verify that it works.

Each time a program accesses the console, it must authenticate itself to the console as a user. Note that if it uses the same login as an existing user, that user is logged out of the system.

Note

Do not use a user provided by LDAP/Active Directory authentiation. Create the user in the Management Console.

Services Layer Clients

Services Layer clients provide an abstracted interface to Build Forge.

The Services Layer is an abstraction between clients and Build Forge internals. The following sections describe how to download and set up clients for the Services Layer.

Java Client

Use the Java Client to write Java programs that access the Management Console.

JDK 1.5 is required for use with the Java API. To use the Java API, you must:

Document # SC23-7681-07

- Get the client API package from your Management Console machine
- Put the .jar on your CLASSPATH

Note

A Build Forge user must be defined on the Management Console for programs to use to authenticate.

You can use the Java API to create Java programs that run on a client machine and access data on the Management Console. The Java API consists of a jar containing classes that define Management Console objects methods that provide operations on those objects.

Documentation is provided in JavaDocs.

Getting the Java Client

You can download the compressed Java client software package from your Management Console host.

To download the Java API, do the following:

1. Access the Client download directory.

   In a web browser, access the following URL:

   `http://<hostname>:<portnumber>/clients/`

2. Save the JAR file.

   Under Java Client, right-click the **JAR file** link and choose **Save Link As** . Specify where to save the JAR file.

3. Save the JavaDocs.

   Under Java Client, right-click the **JavaDoc reference ZIP** link and choose **Save Link As** . Specify where to save the JAR file.

You can access the documentation through the Management Console. On the Client download directory page, under Perl Client, click **JavaDoc reference** .

Setting Up the the Java Client

Place the Java API on a client machine and set up the JDK to use it.

The machine will act as a client to the Management Console machine.

1. Place the jar file where you want it.

2. Update your CLASSPATH.

   Set your CLASSPATH to include the directory where you placed `rbf-services-client-java.jar`.

Document # SC23-7681-07

Perl Client

Use the Perl Client to write Perl programs that access the Management Console.

To use the Perl Client, you must:

- Get the Perl Client package from your Management Console machine
- Install the package (along with Perl if it is not already installed)

Note

A Build Forge user must be defined on the Management Console for programs to use to authenticate.

You can use the Perl Client to create Perl programs that run on a client machine and access data on the Management Console. The Perl Client is a set of Perl modules which provide access to an abstraction of Management Console data objects and methods.

Documentation for Perl Client modules is included inside the client API package in two forms:

- A file, `apidoc.txt`
- Perl POD documentation. For more information, see the online documentation at http://www.perl.org [http://www.perl.org/].

Getting the Perl Client Package

You can download the compressed Perl Client software package from your Management Console host.

To download the Perl Client, do the following:

1. Access the Client download directory.

   In a web browser, access the following URL:

   `http://<hostname>:<portnumber>/clients/`

2. Save the ZIP file.

   Under Perl Client, right-click the **ZIP file** link and choose **Save Link As** . Specify where to save the ZIP file.

3. Save the documentation.

   Under Perl Client, right-click the **PerlDoc reference tar.gz** link and choose **Save Link As** . Specify where to save the ZIP file. Unzip the file to access documentation for each module.

You can access the documentation through the Management Console. On the Client download directory page, under Perl Client, click **PerlDoc reference** .

Document # SC23-7681-07

Setting Up the Perl Client

To use the API, you must install the client API software on a host where you plan to run your applications.

The host will act as a client to the Management Console host.

1. Install a Perl interpreter on the client host, such as ActiveState's ActivePerl version 5.8.4 or later.

   The following Perl prerequisite modules are required (ActivePerl version 5.8.8 includes them):

   - `Exporter`
   - `LWP::UserAgent`
   - `HTTP::Request`

   See the Perl documentation for information on installing Perl modules.

2. Uncompress the Perl Client package (which you downloaded previously) to a temporary directory.

3. Install the Perl Client as a standard Perl distribution as described in the `apidoc.txt` file.

   On Windows you need nmake, which is included in Visual Studio or downloadable from the Microsoft web site (nmake 1.5–it must be installed where it can be found by the PATH environment variable, such as `C:\Windows`).

   In the temporary directory where the Perl Client package was uncompressed, execute these commands:

   ```
   perl Makefile.PL
   nmake
   nmake install
   ```

   On UNIX or Linux systems (or in a Cygwin environment on Windows):

   ```
   perl Makefile.PL
   make
   make install
   ```

   Once installed, the top Perl client module is `BuildForge::Services::DBO`. See the PerlDoc for each module for more information.

Perl API

The Perl API supports programs written in previous versions of Build Forge.

Important

The Perl API is deprecated. Support for it will be discontinued in the future. You should migrate existing programs to use the Perl Client for the Services Layer.

To use the Perl API, you must:

Document # SC23-7681-07

- Get the client API package from your Management Console machine
- Install the package (along with Perl if it is not already installed)

Note

A Build Forge user must be defined on the Management Console for programs to use to authenticate.

You can use the Perl API to create Perl programs that run on a client machine and access data on the Management Console. The Perl API is a set of Perl modules which provide access to Management Console data objects and methods.

Documentation for Perl Client modules is included inside the client API package in two forms:

- A file, `apidoc.txt`
- Perl POD documentation. For more information, see the online documentation at http://www.perl.org [http://www.perl.org/].

Getting the Perl API Package

You can download the Perl API package from the Management Console.

1. Access the Client download directory.

   In a web browser, access the following URL:

   `http://<hostname>:<portnumber>/clients/`

2. Save the ZIP file.

   Under Perl API, right-click the **ZIP file** link and choose **Save Link As** . Specify where to save the ZIP file.

3. Save the documentation.

   Under Perl Client, right-click the **PerlDoc reference tar.gz** link and choose **Save Link As** . Specify where to save the ZIP file. Unzip the file to access documentation for each module.

You can access the documentation through the Management Console. On the Client download directory page, under Perl API, click **PerlDoc reference** .

Installing the Perl API

To use the Perl API, you must install the software on a host where you plan run your applications.

The host will act as a client to the Management Console host.

1. Install a Perl interpreter on the client host, such as ActiveState's ActivePerl version 5.8.4 or later.

   The following Perl prerequisite modules are required (ActivePerl version 5.8.8 includes them):

- `Exporter`
- `LWP::UserAgent`
- `HTTP::Request`

See the Perl documentation for information on installing Perl modules.

2. Uncompress the Perl API package (downloaded in a previous step) to a temporary directory.
3. Install the Perl API as a standard Perl distribution as described in the `apidoc.txt` file.

   On Windows you need nmake, which is included in Visual Studio or downloadable from the Microsoft web site (nmake 1.5–it must be installed where it can be found by the PATH environment variable, such as `C:\Windows`).

   In the temporary directory where the Perl API package was uncompressed, execute these commands:

   ```
   perl Makefile.PL
   nmake
   nmake install
   ```

   On UNIX or Linux systems (or in a Cygwin environment on Windows):

   ```
   perl Makefile.PL
   make
   make install
   ```

   Once installed, the top Perl API module is `BuildForge::API`. See the PerlDoc for each module for more information.

Document # SC23-7681-07

Glossary

This section provides definitions for the concepts and terms used throughout the system.

Access Group

A collection of users who share permissions, notification, and LDAP/Active Directory properties. You can map an access group to an LDAP group, and you can send notifications to a group. You can nest groups. Users inherit the permissions of the groups to which they belong.

Adaptor

An adaptor is an add-on that allows the system to interact with an external system, such as a source control system, debugging database, or testing system. For example, source code adaptors allow the system to monitor and track changes in source control systems such as IBM® Rational® ClearCase®, Perforce, Visual SourceSafe, or CVS, and perform actions based on those changes. The system can use an adaptor to collect information for storage in the Bill of Materials (BOM), or push information back to other information systems.

Agent

A lightweight client component of the Build Forge® system, which must be installed on any machine you want to use as a server in the system. Each agent communicates with the Management Console through the engine to get commands to execute on its server. By default, the agent listens for communications from the engine on port 5555.

Archive

A list of jobs whose output files have been deleted but which still have data in the database.

Built-in Properties Reference

The Management Console collects built-in properties from servers and then assigns the values to the server manifest.

List of Built-in Properties

Built-in properties are used by several different data objects in the system:

- **Selectors** can use built-in properties as selector variables, to match servers with certain values in those properties.
- **Collectors** use built-in properties to collect data from servers.
- **Manifests** store the values of built-in properties when they have been collected.

Document # SC23-7681-07

Built-in properties are not automatically added. You must add a built-in property to a collector for the property to display in the manifest.

Document # SC23-7681-07

Built-in Properties for Collectors and Manifests

Property	Description
CPU_ARCH	The returned value is a `label` for an architecture name, as shown: • `HP-PA`: HP Precision Architecture • `IA-64`: Intel Itanium • `MVS`: IBM S/390 • `PPC`: PowerPC • `PPC-64`: PowerPC 64 • `SPARC`: Sun SPARC • `X86`: x86-compliant architecture used by Intel, AMD, Cyrix, and others.
CPU_LOAD (Windows only)	For Windows systems, the CPU load (or CPU usage) is expressed as a percentage of capacity (between 0 and 100).
CPU_LOAD1 (UNIX/Linux only)	For UNIX/Linux, the average number of processes (load average) running or waiting to run over the last minute. CPU_LOAD1 is a measure of CPU activity. An idle computer has a load number of 0. Each process that is using CPU or waiting for CPU adds to the load number by 1.
CPU_LOAD5 (UNIX/Linux only)	For UNIX/Linux, the average number of processes (load average) waiting to run over the last 5 minutes as reported by the operating system. CPU_LOAD5 is a measure of CPU activity. An idle computer has a load number of 0. Each process that is using CPU or waiting for CPU adds to the load number by 1.
CPU_LOAD15 (UNIX/Linux only)	For UNIX/Linux, the average number of processes (load average) waiting to run over the last 15 minutes as reported by the operating system. CPU_LOAD15 is a measure of CPU activity. An idle computer has a load number of 0. Each process that is using CPU or waiting for CPU adds to the load number by 1.
CPU_MHZ	Processor speed in Megahertz. Certain conditions have to be met for this property to be filled in successfully: • Linux: frequency scaling must be enabled. • Windows: the ~MHz registry entry must exist and be filled in. • x86 and x86-64 processors: inline assembly must work.

Document # SC23-7681-07

Property	Description
CPU_MANUFACTURER	Company name of the processor manufacturer. The names are assumed based on architecture if the information is not directly available. No value is returned if there is insufficient processor information available. Supported values are: • `AMD`: for their x86 and AMD64 processors • `Cyrix`: for their x86-compliant processors • `DEC`: for Alpha and VAX • `HP`: Hewlett-Packard Precision Architecture • `IBM`: IBM S/390 and PowerPC G5 • `Intel`: Intel x86 (including Intel64), IA-64 Itanium • `Motorola`: PowerPC G4 • `NexGen`: x86-compliant processors • `National`: National Semiconductor x86-compliant processors • `Rise`: Rise x86-compliant processor • `Sis`: Sis x86-compliant processor • `Sun`: Sun Microsystems SPARC • `TransMeta`: TransMeta x86-compliant processor • `UMC`: UMC x86-compliant processor • `VIA`: VIA Technologies x86-compliant processor

Document # SC23-7681-07

Property	Description

CPU_MODEL	Manufacturer-specific CPU model numbers. The following values are reported:
	• x86 architecture
	  • 386
	  • 486
	  • 586
	  • 686
	  • X86_64
	• PPC architecture
	  • 6xx
	  • POWER
	  • RS64
	  • G3
	  • G4
	  • G5
	  • Cell

CPU_SERIAL	Serial number of the CPU or machine. Currently this functionality is limited to the following architectures: • x86: Intel or Transmeta serial numbers only. Note: most x86 processors will not report a serial number. No value is returned in those cases. • MacOS/X: the serial number assigned is retrieved from an I/O registry. Requires that CoreFoundation and IOKit frameworks be found.
DISK_FREE	For UNIX/Linux, the amount of free space (in MBs) on the file system specified by the server Path property. For Windows, the free disk space (in MBs) on the drive specified by the server Path property. For example, 4096 MB for systems with 4 GB of free disk space.

Document # SC23-7681-07

Property	Description
DISK_TOTAL	Total free disk space available. This value is reported for the agent's base path, which may have a separate allocation that is smaller than the entire remaining disk or partition. Disk space management varies significantly between operating systems.
MEM_LOAD (UNIX/Linux only)	For UNIX/Linux, the amount of RAM or system memory currently in use is expressed as a percentage of total real memory (between 0 and 100).
MEM_FREE	The amount of RAM or system memory (in MBs). For example, 1183 MB for systems with 2 GB of RAM.
MEM_PAGESIZE	The RAM or system memory page size (in MBs). The standard page size for the host system. For example, 4096 MB is a 4 KB page size.
MEM_TOTAL	The total RAM or system memory (in MBs). For example, 2048 MB for a system with 2 GB of RAM.
NET_FQDN	Fully Qualified Domain Name (FQDN) of the machine where the agent is running. It is reported based on the address that the agent is using to communicate. The address returned can be an IPv4 or IPv6, based on the address actually being used. See also NET_IPV, NET_IPV4, and NET_IPV6.
NET_HWADDR	Hardware address for the interface reported in NET_IFACE.
NET_IFACE	Name of the interface used by the agent. • Windows: the name as reported by the `ipconfig` command, for example `Intel(R) PRO/100 VE Network Connection - Packet Scheduler Miniport` • Other systems: the name as reported by `ifconfig`, for example `en0` or `eth0` or `OSA1`.
NET_IPV	The type of IP connection used to communicate with the agent, either 4 for IPv4 or 6 for IPv6.
NET_IPV4	The IPv4 address used by the agent to communicate. On connections over IPv6, if the agent is able to identify an IPv4 address for the same interface, that address is reported.
NET_IPV6	The IPv6 address used by the agent to communicate.
NET_SPEED (Windows only)	Windows: speed of the interface in Mb/sec, for example `1000` for Gigabit Ethernet.
NUM_CPU	The number of CPUs on the machine.
OS_HOSTID	Result of the `gethostid()` system call. Normally this is not very informative unless a system administrator has set `/etc/hostid` to an informative value.
OS_SYSNAME	The operating system name of the server. For example, Windows XP, AIX, or Macintosh OS, and so on.

Document # SC23-7681-07

Property	Description
OS_RELEASE	The operating system release level of the server. For example, if the server OS is Microsoft XP Version 5.1.2600, this number is 5.
OS_VERSION	The operating system version of the server. For example, if the server OS is Microsoft XP Version 5.1.2600, this number is 1.
WIN_SERVICEPACK (Windows only)	The number of the Windows service pack installed on the server. For example, 2 for Service Pack 2.

Class

Every project and job must be assigned to a class. Classes are groupings of projects which have global properties; most of their properties manage the automatic deletion of outdated project runs generated from projects in the class, based on properties you specify. For more information, see “About Classes” on page 62.

Collectors

You define collectors to determine what properties the system collects from (or assigns to) your servers. The collector assigned to a server is like a blueprint for the server's manifest.

The Collectors section of the Servers module (**Servers** → **Collectors**) lists the available collectors and allows you to create new collectors.

A collector consists of a series of properties which are assigned to any server that uses the collector. However, the specific values of the properties can vary from server to server, because a collector is a set of instructions for collecting data.

You can define several types of properties in a collector:

Set Value — These properties simply assign a named, static value to the server. Specify the property name and the value. The same value is assigned to all servers that use the collector.
The system recognizes values with special names as defining behavior for a server. These special values begin with the underscore (_) character. See “Special Set Value Properties” on page 27 for a list of these.

Built-in — These properties return information about the server. For a list of built-ins, see “Built-in Properties Reference” on page 226. When you choose this type, the system allows you to pick a built-in from a list.

Run Command — For these properties, you define a command for the system to run. The command's output is used to populate the property value in the manifest. You can use a regular expression to specify what part of the command's output to use.
To define a Run Command property, fill out the following fields:

	Property	The name of the property, for the manifest.
	Command	A command to run on the server.
	Regular Expression	A Perl regular expression. Optional. If specified, the system tries to match the regular expression against *each line* of output from the command. The first time a line matches, it retrieves the value of $1 (a Perl convention), and uses it as the value for that property. The regular expression must include at least one set of parentheses so that it returns a value. Consult Perl documentation for more information on constructing Perl regular expressions.

Note

If you do not specify a regular expression, the system uses *up to the first 255 characters* of output from the command to populate the property in the manifest.

.include	These allow you to nest collectors. When you create a .include property, you specify the name of another collector as its value. When it creates or updates the manifest, the system inserts the properties from the referenced collector.

Note

The system applies collector properties in the order they are listed in the collector, and later properties of the same name *override* earlier ones. Use this feature when you include one collector within another one. If you want to use some of the properties of a collector but not all, override the ones you do not want to use.

The system also applies a few properties automatically, such as the BF_NAME property that contains the logical name of the server. See “Special Manifest Properties” on page 26.

Database

The database stores all the information entered into the Management Console as well as data created by the system when it runs a project or logs user actions.

Engine

The engine uses information entered via the Management Console and stored in the database to control project execution, send notification e-mails, and communicate with agents (running on servers).

Document # SC23-7681-07

Environments and Environment Variables

An environment is a container for a set of environment variables. Use environments to create sets of reusable settings that projects, steps, and servers can reference. When the system constructs the environment for a step, it applies the server, then the project, and then the step's environment to create a composite environment. For more information on how to use environments, see “About Environments” on page 31.

Job

An instance of running a project. Typically, a job generates multiple output files on a server. The system stores log and statistical data for each job in the database.

Management Console

The component of the system that is installed on a single machine to coordinate the system. You log in to the Management Console to start or schedule project runs and to view results and reports. The Management Console issues instructions to agents to complete jobs.

Manifests

A *manifest* for a server is the set of values that the Management Console collects from or assigns to that server and stores as a record in the database.

When it chooses a server for a project or a step, the system compares a selector against its set of manifests and picks a matching server.

You can view the manifests for your servers in the Servers module. Select **Servers** → **<server_name>** , then click the **Manifest** tab.

You cannot directly change the manifest for a server. Instead, you must edit the collector assigned to the server, or assign a different collector to the server. The collector defines the kinds of properties the system assigns to a server or attempts to collect from it; the manifest is the resulting set of property values.

MD5

MD5 is a hash algorithm that produces 128-bit numbers to represent a message; these can serve as a digital signature. The system uses MD5 values as unique identifiers for specific versions of a file; the system generates MD5 values for files, and you can search on MD5 values.

Notification Templates

A notification template defines the content and format of the e-mail sent to an access group on the occurrence of a specific event. The system comes with many default templates; you can edit these and create new ones that are a specific to a particular project. See “About Notification Templates” on page 65.

Document # SC23-7681-07

Plug-ins

The plug-ins provide the ability to access your system via an integrated development environment (IDE). For more information, see “Managing Licenses” on page 172.

Project

A project is the system's term for a set of one or more actions which can be launched on servers in the system. Each action is called a step, so a project can be seen as a container for steps. Some properties are set at the project level; some of those properties are inherited by the steps, which can also override them. You can assign environment variables to a project. Every time you run a project, the system stores data about the run as a new job data object.

Run queue

The run queue for the Management Console is the maximum number of currently running jobs.

Selectors

Server selectors allow you to describe the kind of server that a project or step should use by listing desired properties and values. When you apply a selector to a project or step, the system uses the selector to determine which servers are valid choices for the task, and then selects an available server from the valid ones.

You can use selectors to be specific (choosing a specific server by name) or general (any Windows® server). A selector is a list of properties which describe the desired server. To manage selectors, use the **Servers** → **Selectors** page.

If you want to select servers based on properties you define, create appropriate collector properties *first*. For example, you can create a collector with a property named BUILDING (with a value equal to the name of the building that houses a server). This allows you to select servers based on their physical location. However, when you create selector variables, you must choose properties from a list that the system generates from all of your collectors.

If a selector *does not* find a server that matches its property list, then the project or step fails and the system creates a build note.

A selector contains a list of property/value pairs called *variables*. For each variable, you can specify a value and a comparison. For example, you could specify a property "CompilerVersion = 1.1" to select only servers that have that property, but you could also specify "CompilerVersion >= 1.1" to select servers with versions 1.1, 1.3, 2, and 2.0. Selectors support numeric and string comparison operations.

- A variable can be required or optional. When multiple servers match the required variables, the system picks the one that matches the most optional variables.
- You can repeat optional variables in a selector, to increase the score of a server that matches them. For example, you might require MEM_TOTAL> = 1GB but repeat MEM_TOTAL >= 2

Document # SC23-7681-07

GB three times to bias the system to choose servers with memory of at least 2 GB. See below for the details of how the system makes its choice.

- You can use the .include statement to add environment variables from another selector. The .include statement references an environment. If there are duplicate variables in environments, the system counts each instance of the variables when it assigns scores to servers.

To choose a server, the system:

1. Compiles a list of the servers which contain all the *required* variables in the selector.
2. Rates each server, granting the server a point for each *optional* variable it matches.
    - If the selector contains more than one copy of the same variable, the extra copies grant extra points to servers that match them.
    - The system assigns one extra point to the server with the lowest BF_LOADRATIO value.
3. Chooses server that received the most points.

Although it does not appear in a server's manifest, the property BF_NAME is automatically assigned to every server; its value is the logical name of the server. To create a selector for a specific server, create a selector variable that selects BF_NAME=<*logical server name*>.

When comparing selector variables with manifest properties to look for matches, the system looks at the variable's value and the manifest property's value. The system performs a lexical (string) comparison unless **both** values match the following criteria for numbers:

- If the value starts with a digit and contains only digits and decimal points followed by at least one digit, the system performs a numeric comparison.
    - 5, 5.5, 0.5, 5.0, and 5.5.5 are considered numbers.
    - 5., .5, 5.., 5..5, 5.4.6_05, and 5.6i5 are all considered strings
- A numeric value containing more than one decimal point causes a sub-version numeric comparison, where the system compares each decimal-separated section. While 5.21 is less than 5.3 (ordinary numeric comparison), 5.21.0 is greater than 5.3 (sub-version numeric comparison).

Note

The system always performs a string comparison if you use the “contains”operator. Also, the “contains” operator is case-insensitive.

Document # SC23-7681-07

Selector/Manifest Comparison Examples

Property Name	Manifest property value	Operator	Selector Variable Value	Comparison Type	Match?
PerlVersion	v5.8.4	>=	5.2.1	Lexical	Yes
PerlVersion	v5.8.4	>=	v.5.2.1	Lexical	Yes
PerlVersion	v5.8.4	>=	v5.22.1	Lexical	Yes
OS_VERSION	1.15	>=	1.1	Numeric	Yes
OS_VERSION	1.10	>=	1.1.0	Sub-version numeric	Yes
BF_NAME	WinServer1	contains	win	Lexical	Yes
BF_NAME	Server123	=	123	Lexical	No

Semaphores

Semaphores are global flags you establish to prevent activities from occurring at the same time. Because they are global, you can use them to manage processes across projects. Each semaphore amounts to a label that the system manages; a step can get a semaphore by name, and while one project has the semaphore, no other project can get it. A step can also put a semaphore back into availability, but when a project ends, the system also automatically releases any semaphores that the project used.

See “Semaphores” on page 119 for more detail on how to use semaphores.

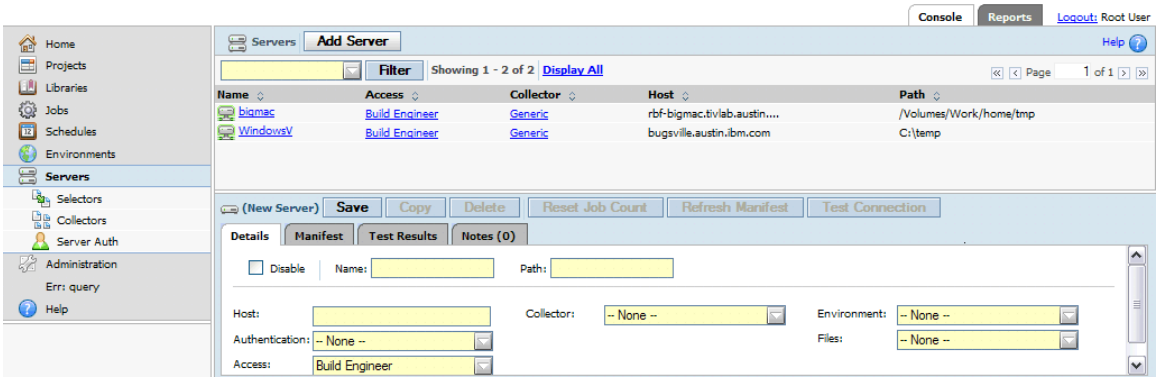
Servers

In Build Forge, a server is a machine that is set up so that the Management Console can run steps on it.

To set up a machine to be available as server in Build Forge, you must do the following:

- Install an agent on the machine (see the *Build Forge Installation Guide* for more information)
- Create a Server resource using the Management Console

The Servers Module



Document # SC23-7681-07

- Servers have manifests. A *manifest* is a list of server properties. A manifest is populated when a collector runs. If a server does not have a collector assigned to it, a few properties are automatically populated in the server's manifest.
- Manifests are populated by collectors. A *collector* is assigned to a server. A collector both sets property values and collects values for properties from the agent for a Server.
- Projects can use selectors to determine what server will run a step. A *selector* reads server properties from the manifest.

As an administrator setting up the system: first, you create servers. You then create collectors that you can assign to servers. You run the collectors to populate the server manifests. Once that is done, build engineers can create projects that use selectors to determine where project steps run.

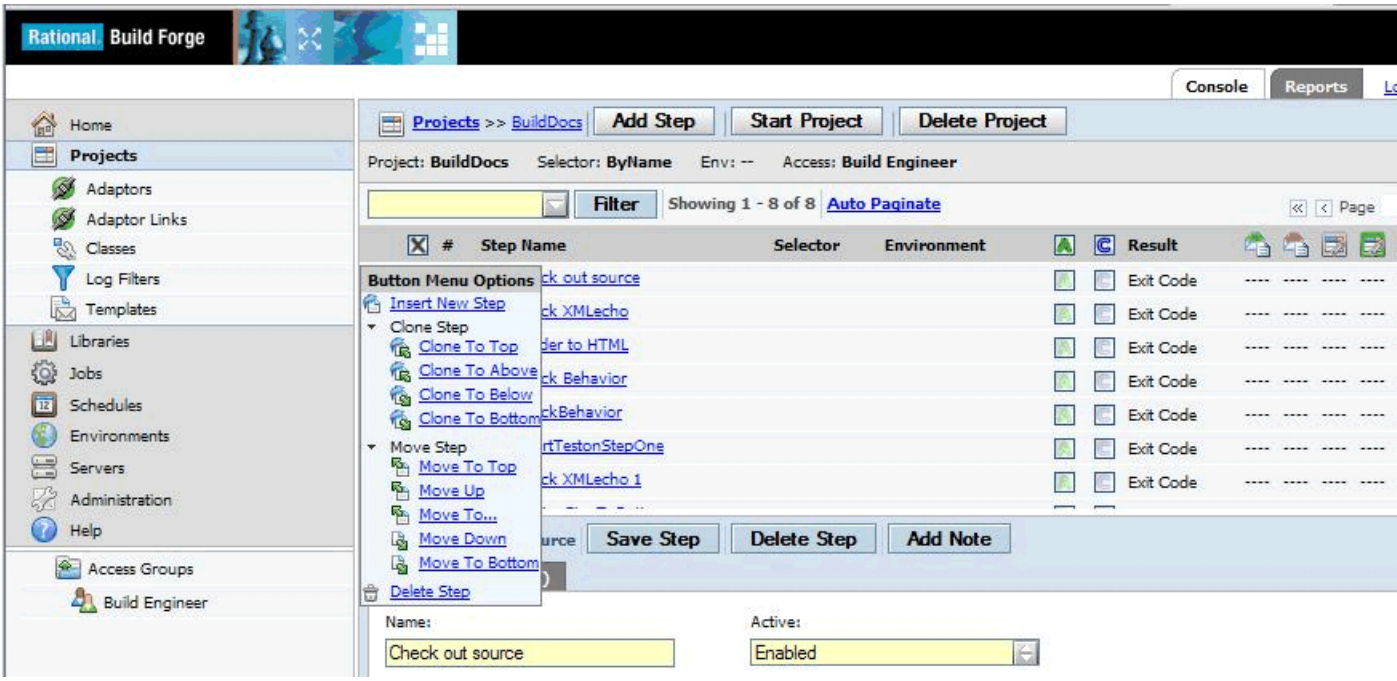
Steps

A step is a component of a project. It contains one or several command lines which can be executed on a server. You can specify properties for a step to tell the system how to handle its output, whether to thread the step, and what to do if the step fails. You can also launch other projects from a step.

To work with steps, select **Projects** , then click on a project name. The system displays the list of steps for the selected project.

- To edit a step, click the step name. The system displays the step's properties in the lower pane. Make your changes, then click the **Save Step** button.
- To add a step to the end of the project, click the **Add Step** button at the top of the main pane. The system displays an empty step details form. Enter values for the properties (you must enter a **Name** and **Command** at least), then click the **Save Step** button.
- To insert, move, copy, or delete a step, click the  Actions icon at the beginning of the step's row in the main pane to display a pop-up menu and select an appropriate option.

Document # SC23-7681-07



Note

Use the Clone Step commands to copy a step. These commands create a new step, inserted before or after the original step, with the same name and an appended number. For example, if you clone the step "Make Driver", the system inserts a copy named "Make Driver 1." A cloned step has the same properties as the original.

- To disable a step, click the ☐ **Disable** icon that precedes each step. A check mark ☒ indicates that the step is disabled. Alternatively, disable a step by using the Action field in the **Step Details** form.

  A disabled step is not used when the system runs the project; it does not appear in the list of available steps if a user manually starts the project and selects specific steps to include or exclude. Use this feature to prevent an unfinished step from being executed, or to temporarily remove a problem step from a project.

Timeout

Each step has a timeout property that specifies how long the system should wait for the current command to produce output. If the timeout value is reached, the system fails the step.

The timeout property takes a value in one of the following formats, converting all formats to a value expressed in <minutes>:<seconds>.

6:20 (6 minutes, 20 seconds)

720 (720 seconds)

14m (14 minutes)

Document # SC23-7681-07

Whenever a command stops producing output, the system starts a timeout counter. Every time the command produces output, the system resets the counter. For example, if the timeout is set to the default value of 5:00, the command could run forever so long as the command created a line of output every 4 minutes and 59 seconds.

You can override the Timeout property of a step by declaring a _TIMEOUT environment variable. When a variable of this name exists as part of the step's environment (at the step level, or inherited from the project or server), the value of the _TIMEOUT variable replaces the Timeout property value.

Note

If the Timeout value is zero, the step is allowed to run indefinitely.

Threading

To thread a step is to set its properties so that it can run in parallel, thereby increasing the speed of your project. To make a step threadable, set its **Thread** property to Yes. When a series of steps are threadable, the system sends each step out to a server without waiting for the first one to complete.

User

The system maintains its own set of users and permission settings. You can define what users are allowed to do by making the users members of access groups with specific permissions, and you can set individual user preferences. For more information, see “Creating and Editing Users” on page 155.

Document # SC23-7681-07

Contacting IBM Customer Support for Rational Products

If you have questions about installing, using, or maintaining this product, contact IBM Customer Support as follows:

The IBM software support Internet site provides you with self-help resources and electronic problem submission. The IBM Software Support Home page for Rational products can be found at http://www.ibm.com/software/rational/support/.

Voice Support is available to all current contract holders by dialing a telephone number in your country (where available). For specific country phone numbers, go to http://www.ibm.com/planetwide/.

Note

When you contact IBM Customer Support, please be prepared to supply the following information:

- Your name, company name, ICN number, telephone number, and e-mail address
- Your operating system, version number, and any service packs or patches you have applied
- Product name and release number
- Your PMR number (if you are following up on a previously reported problem)

Downloading the IBM Support Assistant

The IBM Support Assistant (ISA) is a locally installed serviceability workbench that makes it both easier and simpler to resolve software product problems. ISA is a free, stand-alone application that you download from IBM and install on any number of machines. It runs on AIX, (RedHat Enterprise Linux® AS), HP-UX, Solaris, and Windows® platforms.

ISA includes these features:

- Federated search
- Data collection
- Problem submission
- Education roadmaps

For more information about ISA, including instructions for downloading and installing ISA and product plug-ins, go to the ISA Software Support page.

IBM Support Assistant: http://www.ibm.com/software/support/isa/.

Determining the Management Console Version Number

You can find out what version of the Management Console you are working with by placing your mouse cursor over the logo at the top of the page. The system displays the version number in a pop-up tool tip.

Document # SC23-7681-07

Notices for IBM Rational Build Forge

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

```
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.
```

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web

Document # SC23-7681-07

sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

```
IBM Corporation
Department BCF
20 Maguire Road
Lexington, MA 02421
U.S.A.
```

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Document # SC23-7681-07

Trademarks

IBM, the IBM logo, Rational, ClearCase, ClearQuest, and DB2 are registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published.

Other company, product or service names may be trademarks or service marks of others.

For other IBM trademark attributions, see http://www.ibm.com/legal/copytrade.shtml

Document # SC23-7681-07