

This module covers the basics of Abstracting the Process for IBM Rational Build Forge Version 7.0 and above.

This module assumes users are familiar with IBM Rational Build Forge basics. For a primer on Build Forge, exit this module and first review the Introduction to Build Forge module, then continue with this more advanced topic.

## Objectives

- To understand and implement Build Forge projects
- To understand and implement steps
- To match current processes to Build Forge project implementations



This module discusses Build Forge projects, along with how to implement their steps. This module also shows how to match the current process of the organization into an equivalent process in Build Forge.

## Build Forge projects

- Projects are the logical units for organizing processes.
- A project needs:
  - ▶ Steps
  - ▶ Hardware to run on
- Projects are the units that are picked up and run by the engine.



Projects are the logical units that Build Forge uses for handling processes. This module details the steps that the project goes through, along with detailing how to define a selector that explains where to run those steps. Additionally, the Build Forge engine only works with Projects, thus this module also explains how units are passed to the engine and executed.

## Build Forge libraries

- Libraries are very similar to projects, except they do not have hardware information regarding where to run.
- Libraries are designed for use similar to a function in code.
- If there is a set of operations shared across projects, use a library to reuse the information.



Libraries are almost identical to Projects. The only difference between the two is that the Library does not define a selector. Therefore, the Library does not have any hardware information as to where it should run. The Library is dependent on a Project to give it the hardware execution information, and is used like a function call in code. Typically, there might be a sub-process shared across multiple processes, such as with a checkout procedure. Rather than implementing it in each process that does a checkout, call the library in each process that needs it. This way, it avoids duplicating work. If there is an update to the checkout process, the Build Forge process can be updated in one place.

## Deeper into projects

- Projects/Libraries have these settings:
  - ▶ Name – Logical name of the Project/Library
  - ▶ Max Threads – The number of threaded steps that can run at once
  - ▶ Run Limit – The number of builds that can run concurrently
  - ▶ Pass/Fail Chain – Conditional project/library execution-based project results
  - ▶ Start/Pass/Fail Notify – Conditional e-mail notification settings



The Project/Library has several configuration settings.

The **Name** field defines the Build Forge logical name for the Project or Library.

**Max Threads** is where to define the most concurrent steps that can run at one time. The Project/Library Max Thread value caps the number of threaded steps that run simultaneously, no matter how many threaded values can run at a given moment.

**Run Limit** controls the number of instances that the Project/Library can run at one time. Use this setting to allow one Project/Library instance to run at a time.

**Pass/Fail Chain** sets up conditional project Builds based on the result of this project as a whole. If, for example, this project were to fail, and a project was defined for the fail chain, that fail chain starts upon this project's failure.

**Start/Pass/Fail Notify** sets up e-mail notifications for events when they occur. If a Start notify group is defined for a project, then whenever a Build of that project is started, that group is e-mailed.

## Deeper into projects

- Projects/Libraries have these settings:
  - ▶ Class – The class of the project
  - ▶ Selector – What the project should run on (blank for libraries)
  - ▶ Environment – What environment to run the project with
  - ▶ Console – Globally distributed development setting
  - ▶ Sticky – Changes the default selector behavior for the project

**Class** controls how long Builds of a particular project should remain on the console. Class is further discussed later in this module.

**Selector** is where the selector for this project is defined.

**Environment** sets up the environment variables for the project Build. It is important to note that the environment variables applied are cumulative with those applied at the server level. Environment is further discussed later in this module.

**Console** is a setting for Globally Distributed Development. Console is further discussed later in this module.

**Sticky** changes the default selector behavior for the project. By default, the project uses the selector for every step, so each step can theoretically run on different agents every time. However, if sticky is set, then the project will reserve the first agent it selects and use that agent for every step. This ensures that the agent is consistent for the entire project.

## Additional project settings

- Tags
  - ▶ Control how the project is labeled when a build is generated
  - ▶ Can auto-increment and combine strings and numbers
- Registers
  - ▶ Store persistent information that remains even between executions of a build
  - ▶ Store and persist files for future runs of the project



Tags are what the project uses to name Builds of a project. Tags are very flexible, and use any number of variables. Those variables can be set to automatically increment when Builds are queued.

Registers are used for persisting information between Builds. Build Forge works to keep each Build a totally separate entity.

IBM Software Group | Rational software

IBM

## Project/Library details

Projects >> Projects

Add Project

Filter

Showing 1 - 41 of 41

Auto Paginate

Page 1 of 1

Project	Tag	Class	Environment	Selector	Access
[RSDC] Back up	BUILD_\$B	Insta_Purge		murata	Build Engineer
[RSDC] Compilgate Project	BUILD_\$B	Insta_Purge	[RSDC] UCMEnv	willvm	Build Engineer
[RSDC] createPVOB	BUILD_\$B	Insta_Purge	[RSDC] UCMEnv	willvm	Build Engineer
[RSDC] removePVOB	BUILD_\$B	Insta_Purge	[RSDC] UCMEnv	willvm	Build Engineer
[RSDC] removeUCMConfg	BUILD_\$B	Insta_Purge	[RSDC] UCMEnv	willvm	Build Engineer
[RSDC] UCMConfg	BUILD_\$B	Insta_Purge	[RSDC] UCMEnv	willvm	Build Engineer
Adaptor	BUILD_\$B	Insta_Purge		Any	Build Engineer
Adaptor Test	BUILD_\$B	Scratch	component	linux(magatron)	Build Engineer
Adaptor Test Copy	BUILD_\$B	Scratch	component	linux(magatron)	Build Engineer
Adaptor Test Copy 2	BUILD_\$B	Scratch	component	linux(magatron)	Build Engineer
bart failchain test	BUILD_\$B	Insta_Purge		any machine	Build Engineer

Projects: Adaptor

Save Project

Copy Project

Delete Project

Clobber

Project Details

Tags

Registers

Name: Adaptor

Access: Build Engineer

Max Threads: Unlimited

Class: Insta\_Purge

Console: Primary

Run Limit: Unlimited

Selector: Any

Start Notify: -- None --

Pass Chain: -- None --

Environment: -- None --

Pass Notify: -- None --

Fail Chain: -- None --

Sticky: Not Sticky

Fail Notify: -- None --

Abstracting the process

© 2008 IBM Corporation

This is the project screen in Build Forge. The envelope columns on the right signify the notifications for the project: Start, Pass, and Fail, from left to right. Moving the mouse over the entries shows a popup that displays the value of that setting. The green play button on the left side allows users to start the project with all default settings. Clicking the link under the project column links directly to the steps for that project. To edit the project, click the pencil icon to the left of the project name.



## Build Forge steps

- The smallest logical unit of the process
- Define a single action for the process defined by the project
- Are semantically similar to Projects/Libraries



Steps are where the smallest actions for a process are defined in Build Forge. From a settings standpoint, the step is not very different from the project or the library; it instead defines a smaller, more precise action.

## Step settings

- Steps have these settings:
  - ▶ Pass/Fail Chain, Start/Pass/Fail Notify, Class, Selector, Environment – Same as projects
  - ▶ Active – Shows whether the step is enabled or disabled
  - ▶ Inline – Inserts the steps from the given Library/Project to the current project below this step
  - ▶ Threaded – Determines whether this step can be threaded or if it should be joined
  - ▶ Directory and Path – Where to run the step - path can be relative or absolute to the server path



Steps have certain configuration information:

The **Pass/Fail Chain, Start/Pass/Fail Notify, Class, Selector and Environment** are all the same as the project, only now defined for the step. Note that the environment is still cumulative, so there are potential entries from the server, project, and step. Also, the Selector entry overwrites the Project Selector, allowing a step to select from a completely different pool than all other steps in the project.

**Active** defines whether a step should be run or not when a Build is started.

**Inline** allows a user to insert a project or library's steps below the step with the inline defined. This is similar to the chain, and differences are further discussed later in this module. The inline will receive this project's environment, and is able to make changes to it.

**Threaded** defines if this step can be run concurrently with other steps. There are three options: Yes, No, and Join. The Join option runs with the threaded steps above it, but does not allow the subsequent steps to run until all the threaded steps above it are complete.

**Directory and Path** define the location on the agent to run this step. Directory is the directory path, and path defines if the directory is Relative or Absolute. The common misconception is that Absolute takes the path out to the root directory on the agent machine, which is not true. Absolute only takes the path out to the directory defined in the Server definition of the agent. If the Server is defined with a path of C:\Builds, checking Absolute here goes to that path, NOT to C:\.

## Step settings

- Steps have these settings:
  - ▶ Timeout – Time in seconds to timeout
  - ▶ Result – How to determine pass or failure
  - ▶ Broadcast – Runs on all servers matching the selector instead of selecting a single machine
  - ▶ On Fail – The action to take on failure - halt or continue
  - ▶ Pass/Fail Wait – Only applies to the chains, determines whether the step should wait for the chained project to end

**Timeout** defines how long Build Forge waits for output before failing a step. Note that the step timeout is clocked from the last time any output was received. It is not based on absolute time that a step has run, but from the last time ANY output was received.

**Result** defines how a pass or fail is determined for this step. By default, this is done through exit code. If the step returns a zero exit code, then it passes. Any non-zero code fails. The other option is a log filter, which is further discussed later in this module. It allows a search for specific entries in the step log to cue passes or fails.

**Broadcast** alters the default behavior of the project. By default, the step selects and runs on one Server. However, if Broadcast is set, then the step selects all Servers that fit the Selector criteria, and runs the step on all those Servers simultaneously.

**On Fail** states that the Build should be stopped if the step fails, or if there can be no further progress made.

**Pass/Fail Wait** applies when Pass or Fail chains are defined for the step. It determines that if a Pass or Fail chain has been defined, then the step should wait for that Chain to complete before continuing on to the next step.

## Core of the step

- The last setting is the command field.
- Command field defines a command that runs on the agent on whatever shell is configured for that agent.
- This is the small logical part of the process, and constitutes the building blocks of the Build Forge process.



The final setting for the Step is the command field, where a user defines the actions to take. This is the literal command run on the shell of the Agent selected to run this step.

## Step details

Project: **Adaptor** Selector: **Any** Env: **Build Engineer**

Filter Showing 1 - 3 of 3 [Display All](#)

#	Step Name	Selector	Environment	Result	Access
1	source			Exit Code	Default
2	pass			Exit Code	Default
3	default			Exit Code	Default

Step: **source** [Save Step](#) [Delete Step](#) [Add Note](#)

**Details** **Notes (0)**

Name:  Active:  Access:

Directory:  Path:

Command:

Environment:  Selector:  Broadcast:

Timeout:  Results:  On Fail:

Thread:  Inline:  Pass Wait:

Pass Notify:  Pass Chain:  Fail Wait:

Fail Notify:  Fail Chain:

This slide shows the Step menu for a project. Note that in addition to the Pass/Fail notifies, there are also pencil icons. These define Inline, Pass, and Fail Chain, from left to right. Moving the mouse over these icons shows the values of the settings. When moving the mouse over the far left program icon, a popup menu opens that allows users to move and copy steps. The box column to the right of that icon is the shortcut for disabling steps. If a step has an "X," it is not included in the Build.

## Chained versus inline

- Chained
  - ▶ Fire and forget
  - ▶ Spawns a new build with its own environment
- Inline
  - ▶ Function call
  - ▶ Inserts the library/project into the currently running Build below the step that has the inline



Now that steps have been explained, this module will examine the differences between a Chain and an Inline.

A chained Build is a completely separate Build entity. The chain begins and creates its own copy of the environment. This Build then runs separately and parallel to the currently running Build, and is now unable to affect anything in the original Build directly. This situation is commonly known as “Fire and Forget.” The chaining Build starts it and never thinks about it again, unless Pass or Fail Wait is defined for the step.

Inline is similar to a function call, as everything still occurs in the currently running Build. If a project or library is inlined for a step, it is as if those steps had been cut out from the project or library and pasted below the step defining the inline. Any changes to the environment are reflected in the Build, and those steps appear in the Build.

## Access groups

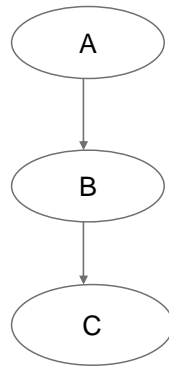
- Control access to Build Forge objects
- Are role based
- Permissions are assigned by access groups, and not by a user.
  - ▶ A user's permission set is determined by the sum of all access groups a user belongs to.
- For any defined access group, there is an option to create child groups.
  - ▶ Child groups inherit all permissions, and can add more.
  - ▶ If the parent group owns a particular Build Forge object, the entire hierarchy of children for that group can see it.



Access Groups allow Build Forge to assert its ownership. If a particular group owns a particular object in Build Forge, such as a Selector, then only that group can view that object. Ideally, Access Groups should be established to match particular roles. A user's particular roles should be assigned to that user, providing access to all objects that a user needs. However, Access Groups also control permissions on the Build Forge console. In addition to assigning object ownership, Access Groups control what actions can be performed on the Management Console. The user's permission set is determined as the sum of all the permissions to all Access Groups a user belongs to.

Access Groups also allow a user to define subgroups, or separate Access Groups, that inherit all permissions from the parent. More permissions can be added to the child, and the parent does not receive those permissions. The child sees all objects assigned to the parent, but objects assigned to the child cannot be seen by the parent. When creating these Access Group hierarchies, they must be carefully planned to prevent unintended inherited visibility and permissions.

## Access group diagram

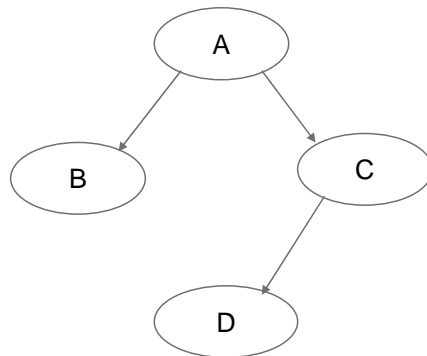


- If a project is owned by group A, then both B and C can see that project.
- If a project is owned by B, C can see the project, but A cannot.
- Both B and C share the same permission set as A. If B adds permissions, C also inherits them, while A does not.

This diagram shows a potential Access Group hierarchy. Access Group C is a child of Access Group B, and B is a child of Access Group A. Permissions pass from A to B to C. Note that if permissions are added to B, then permission set C receives those new permissions. C is ultimately the most powerful class, as it receives everything from the classes above, in addition to any other permissions given to C.



## Access group diagram - complex



- If a project is owned by B, only B can see that project. As far as groups in A, C and D are concerned, that project does not exist.
- If a user belongs to both group B AND D, that user is able to see any project owned by any group A, B, C, or D.

This is an Access Group hierarchy that is a bit more complex. Note that any projects assigned to Access Group B are only visible if a user is a part of group B. However, it is completely possible for a user to be assigned to Access Groups B and D. In this case, a user is able to see everything, as the set of permissions then encompasses all four groups.

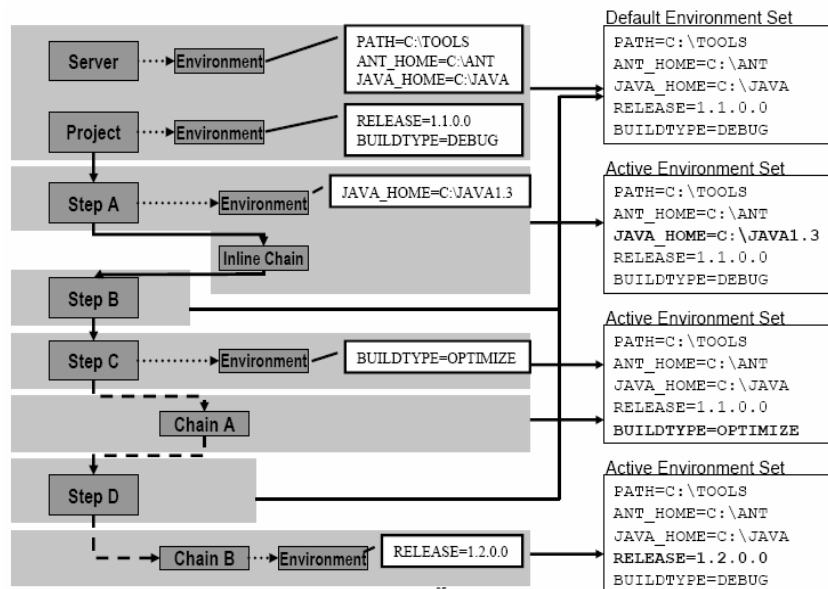
## Environments

- An important part the Build Forge process
- Are cumulative as the build runs
- Servers, Projects/Libraries, and Steps can all have environments.
- Persists as the project is run



Environments are essential for creating the reproducibility and hardware abstraction that makes Build Forge work. The most important thing to remember is that environments are cumulative. Upon changing something in the Build environment, the Build remembers it until it is complete. Everything persists unless Build Forge is given instructions not to continue, or unless the Build terminates.

## Environment hierarchy



This diagram illustrates the flow of a Build and how it affects the environment. First, notice that the Server and Project level environments are immediately put together to constitute the beginning environment set. Each step then adds its environment as the Build runs. For step A, note that the inline chain takes place in the same environment set as the step before it. Step C chains, but the chain receives the environment from the step. At that point, the Build does not care what the chain does with the environment. It is a copy of the step level environment, and this Build will no longer be affected by the other Build. The same base environment that B received is applied to D, but note what the chain is called in D. The environment applied there does not affect the original Build.

## Environment settings

- Environment variables have these settings:
  - ▶ Name – Name of the variable
  - ▶ Value – The value of the variable
  - ▶ Action – Assorted actions that occur when setting up the variable, such as “Delete,” “Append,” “Set if not already set,” and so on.
  - ▶ On Project – Properties associated with the variable, such as “Hide this variable,” “Require this variable to be set,” and so on.



The environment configuration has several settings:

**Name** declares the variable and sets its value.

**Action** defines what to do when the variable is set. There are several options here. Build Forge can delete the variable if it is found, set the variable if it is not already set, can append the value to the end of the existing variable, and so on.

**On Project** defines what should be done on the Build Forge end with a specific variable. Again, there are a few options here: whether Build Forge should require this variable to be set before starting a Build, whether the variable should be visible on the start screen, and so on.

For the full set of options, look at Build Forge help for further reading.

IBM Software Group | Rational software

IBM

## Environment settings

Environments>>[RSDC] UCMEEnv

Add Environment Variable

Filter

Showing 1 - 4 of 4

Display All

Name	Value	Action	On Project
PVOB	BF_PVOB	Set	Normal
PVOB	BF_COMP1,BF_COMP2	Set	Normal
StreamView	BF_TEST_PROJ_INT	Set	Normal
ComponentView	BFVIEW	Set	Normal

ComponentView

Save Variable

Delete Variable

Details

Name:

Value:

Action:

On Projects:

Abstracting the process

21

© 2008 IBM Corporation

This is the Environment setting screen in Build Forge. Examine all available options for the Action and On Project settings.

## Summary

- Projects and libraries are almost the same, except projects have a selector.
- Steps are the smallest logical unit in defining the process.
- Chains are new builds; inlines are added to the existing build.
- Environments are cumulative.



In summary, this module explained various aspects of abstracting the process in Build Forge. Projects and Libraries are the basic unit of process in Build Forge and are largely identical, except Projects define the hardware where they will run. Steps are the smallest unit of process that Build Forge defines, and should be as small as possible when they are defined. Chained projects are “fire and forget.” They receive the environment of the calling step or project, but from then on, the Build is autonomous. The inlines become part of the Build that called them. Environments are cumulative as the Build runs, making any changes made to the environment persist until the Build terminates.

## Feedback

### Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

[mailto:iea@us.ibm.com?subject=Feedback\\_about\\_RBF\\_Module4\\_AbtractingTheProcess.ppt](mailto:iea@us.ibm.com?subject=Feedback_about_RBF_Module4_AbtractingTheProcess.ppt)

This module is also available in PDF format at: [./RBF\\_Module4\\_AbtractingTheProcess.pdf](http://RBF_Module4_AbtractingTheProcess.pdf)



You can help improve the quality of IBM Education Assistant content by providing feedback.

# Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

Build Forge Rational

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at <http://www.ibm.com/legal/copytrade.shtml>

Rational is a trademark of International Business Machines Corporation and Rational Software Corporation in the United States, Other Countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. 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 in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

© Copyright International Business Machines Corporation 2008. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.