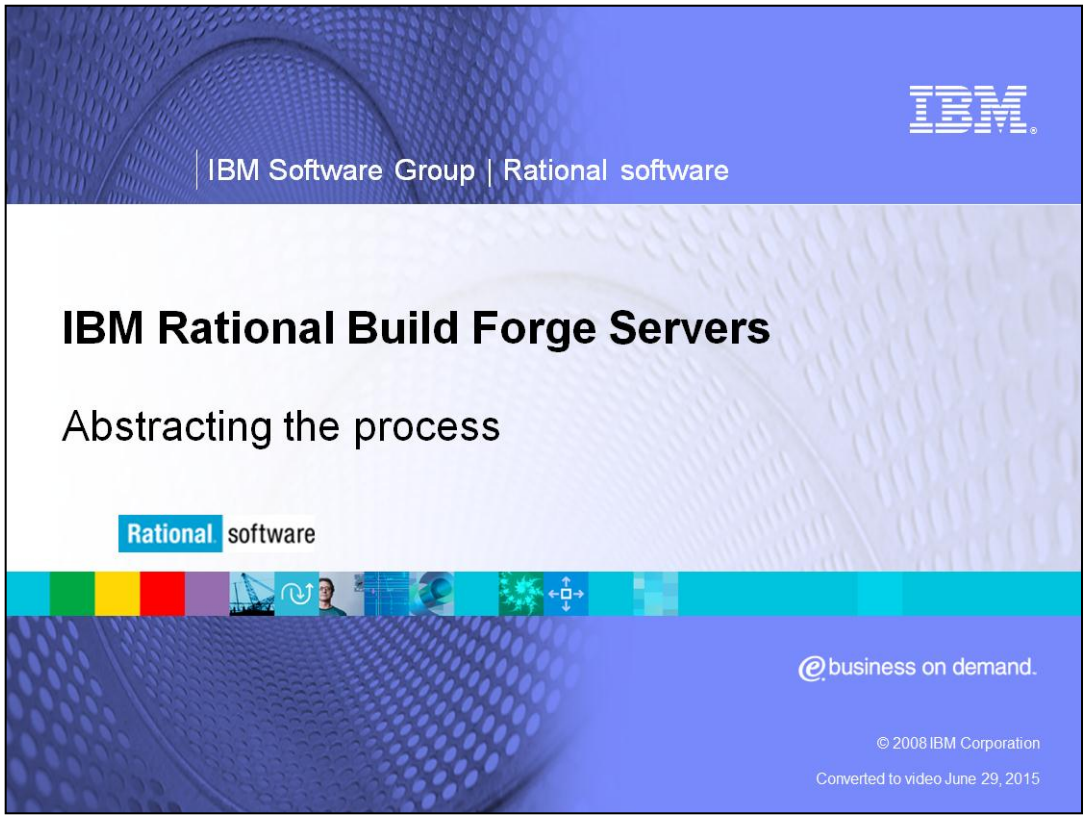
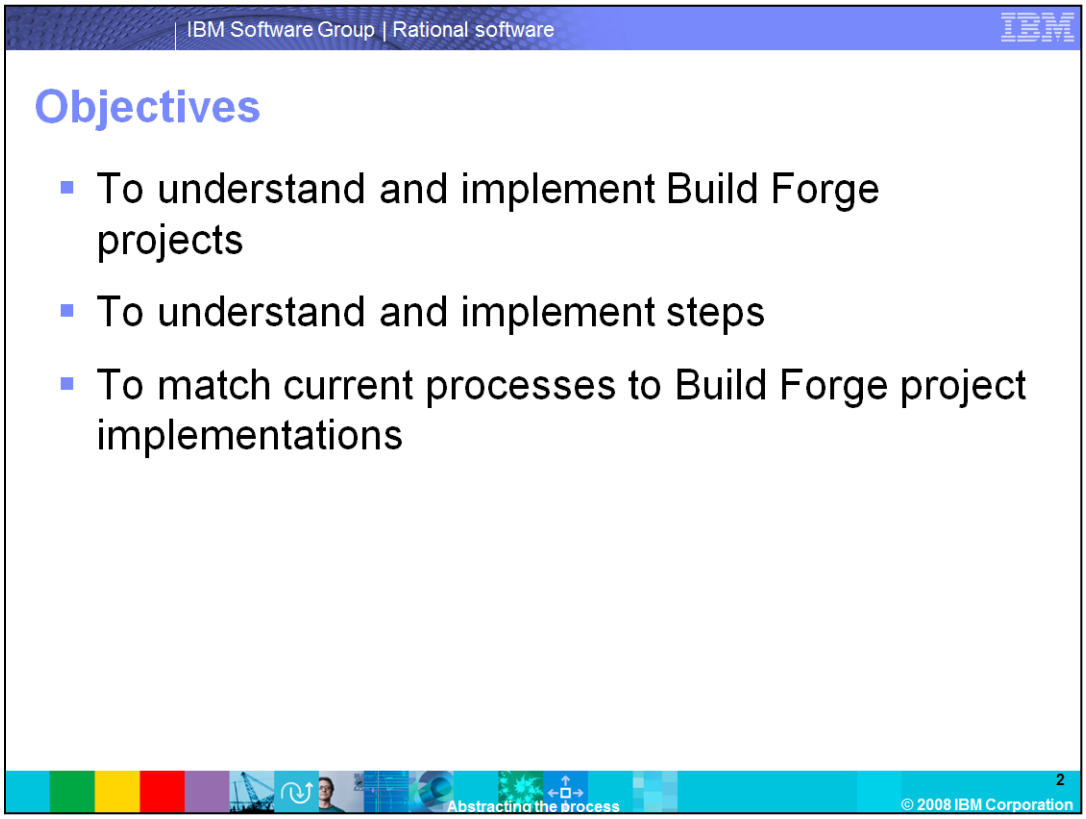


This module covers the basics of Abstracting the Process for IBM Rational® Build Forge® Version 7.0 and above.

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.



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

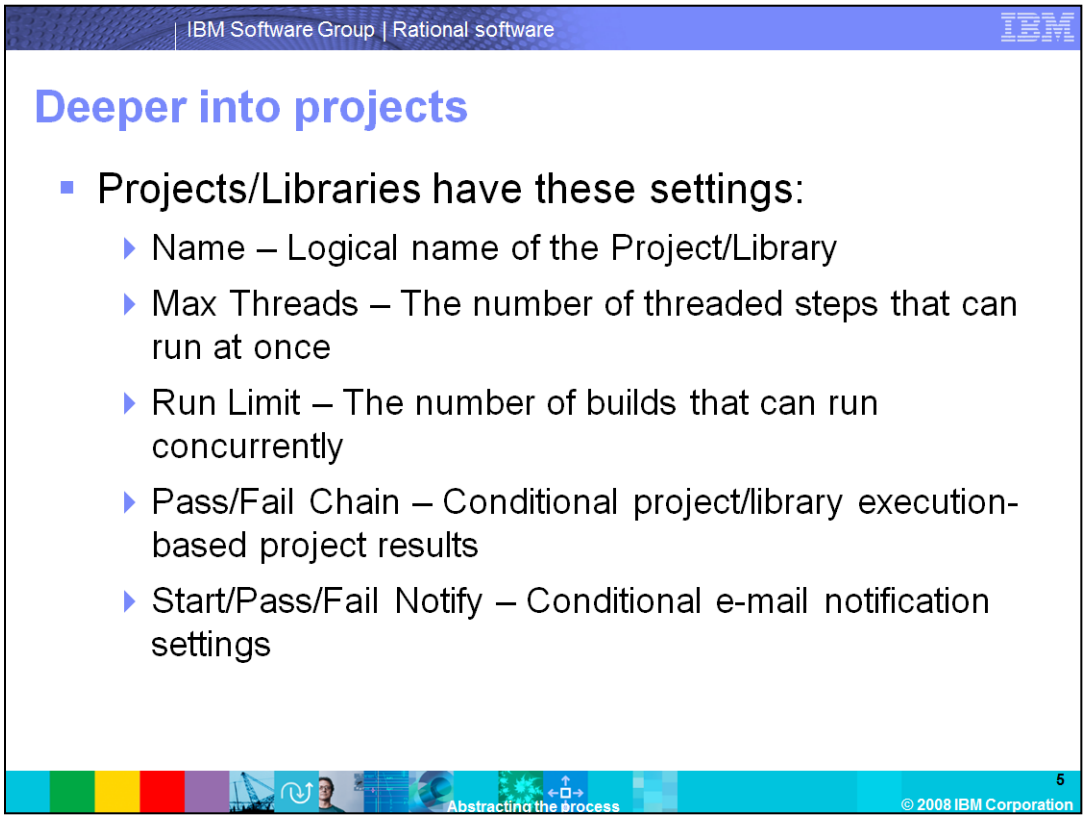
- Projects are the logical units for organizing processes.
- A project needs:
  - Steps
  - Hardware to run on
- Projects are the units that are picked up and run by the engine.

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.



The Project/Library has several configuration settings.

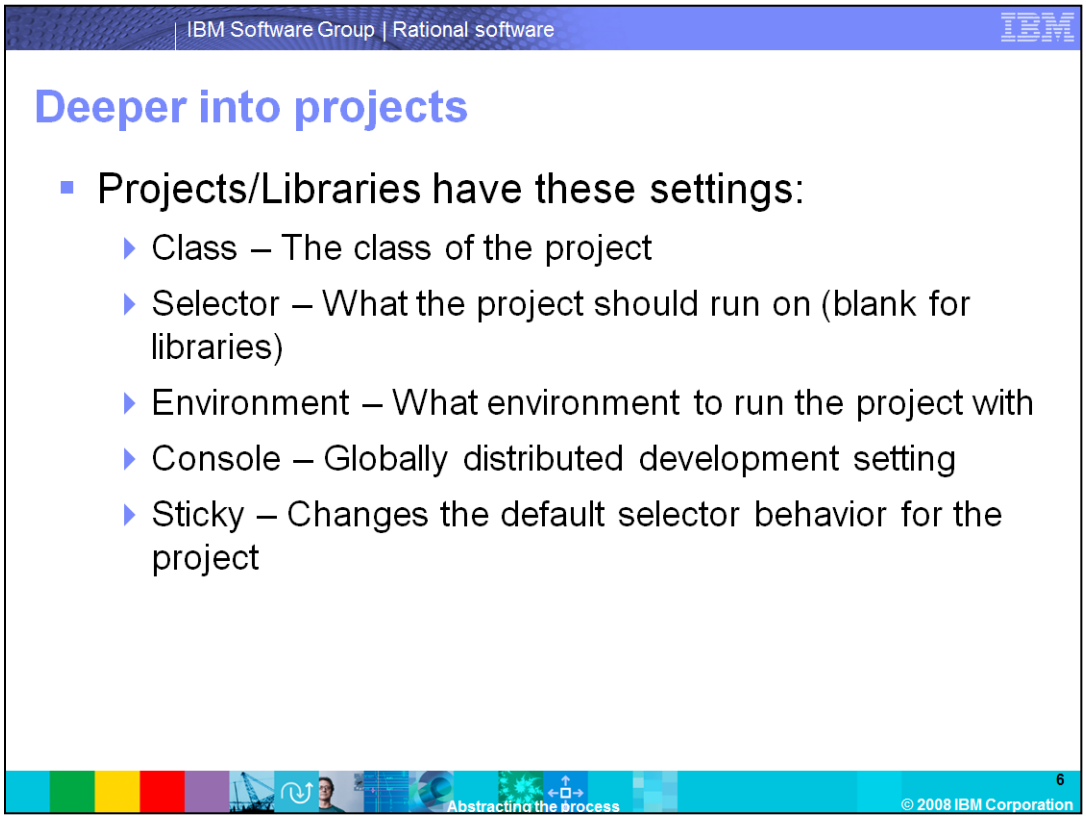
The **Name** field defines the Build Forge logical name for the Project or Library.

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.



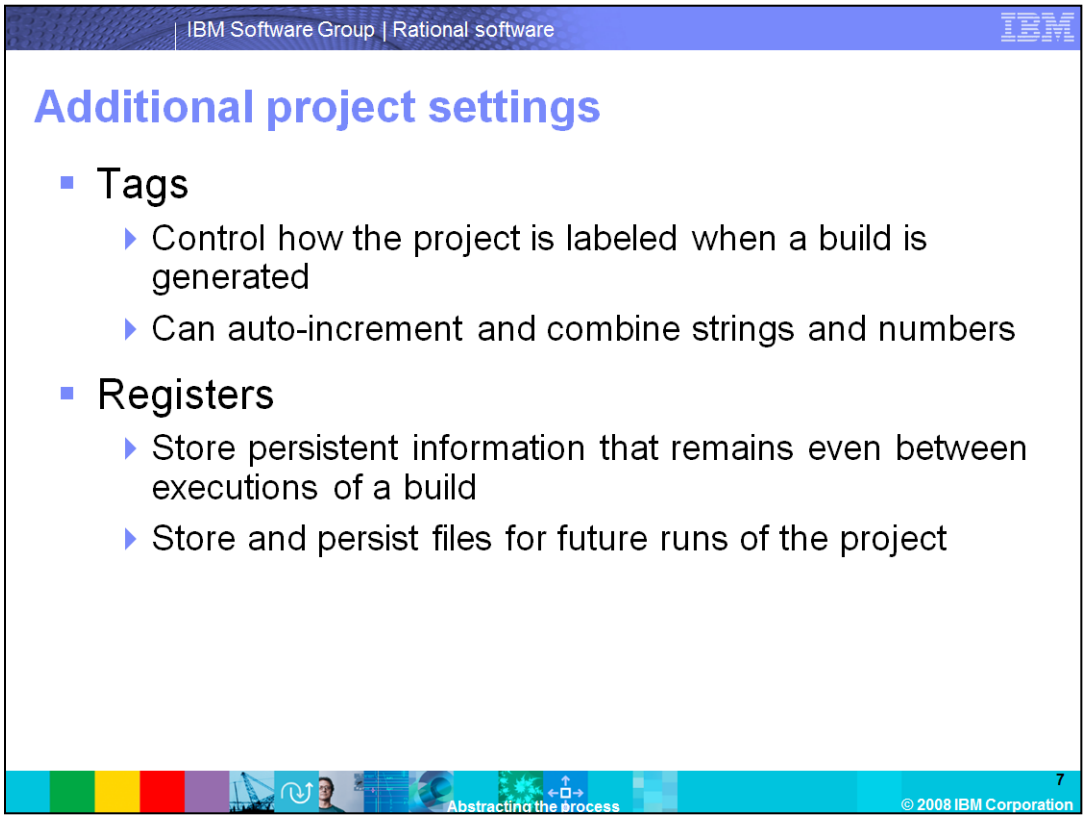
Class controls how long Builds of a particular project should remain on the console. Class is further discussed later in this module.

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.

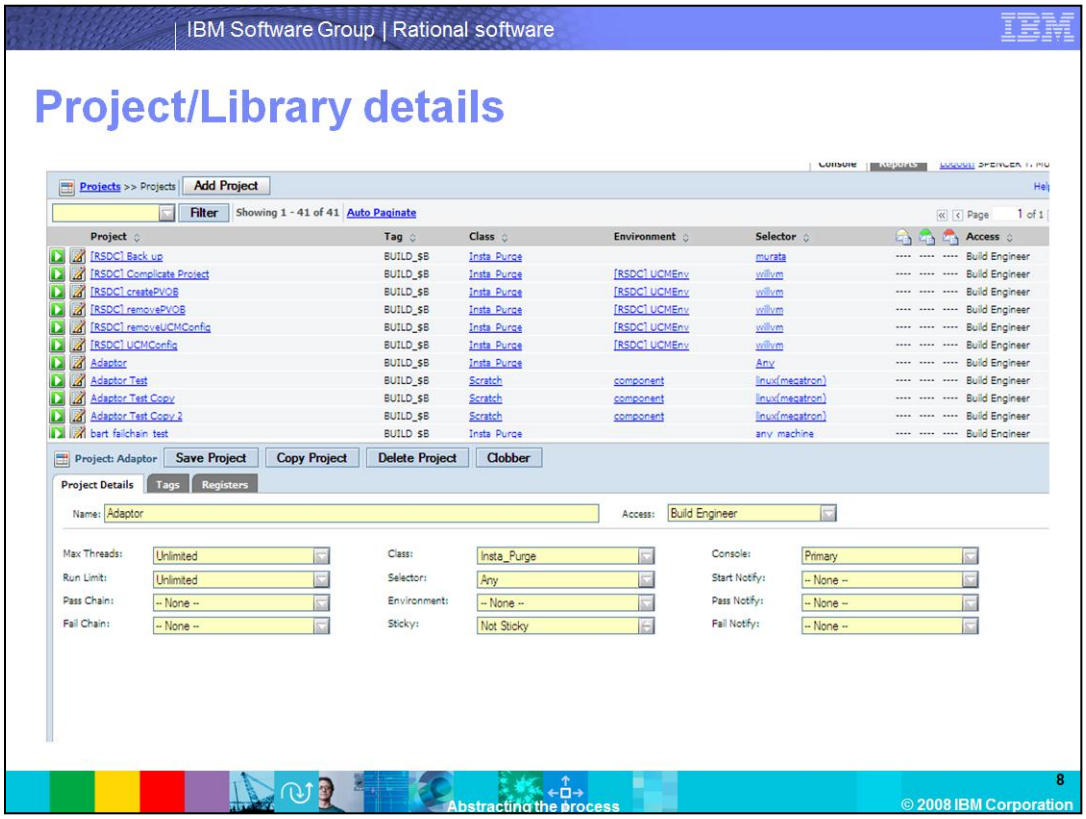
Console is a setting for Globally Distributed Development. Console is further discussed later in this module.

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.



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.

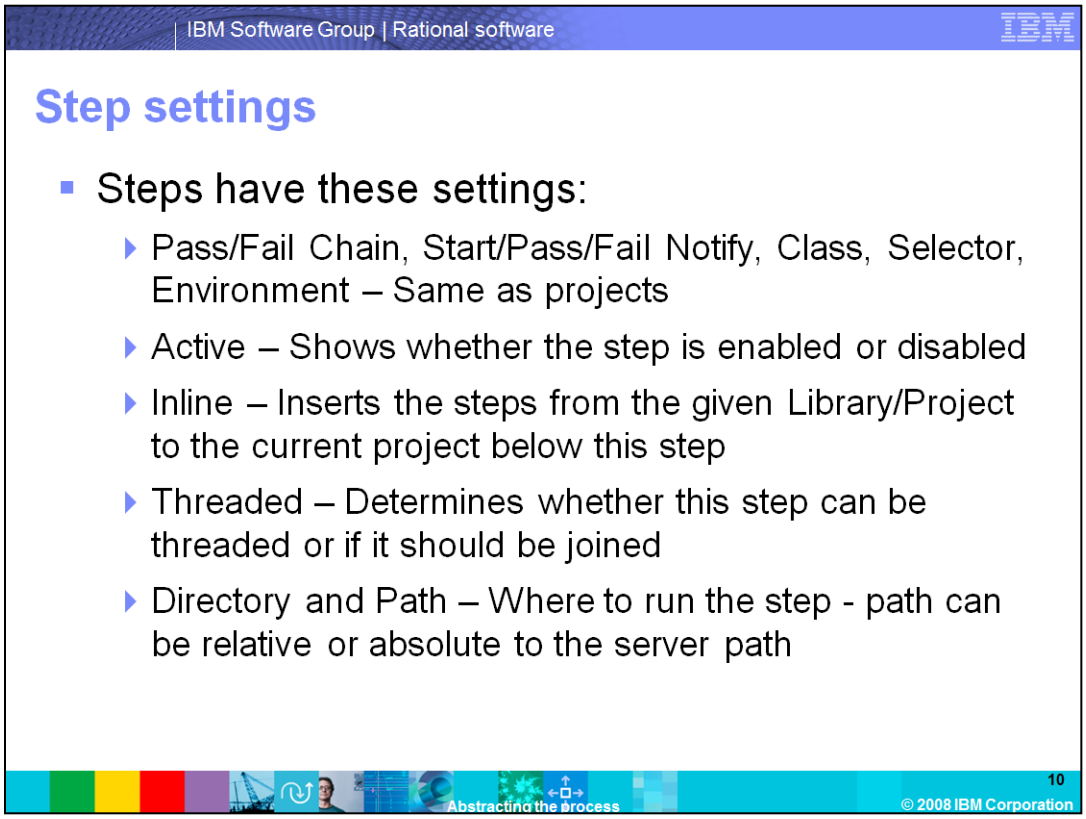


This is the project screen in Build Forge. The envelope columns on the right signify the notifications for the project: Start, Pass, and Fail, from left to right. Moving the mouse over the entries shows a popup that displays the value of that setting. The green play button on the left side allows users 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.



Steps have certain configuration information:

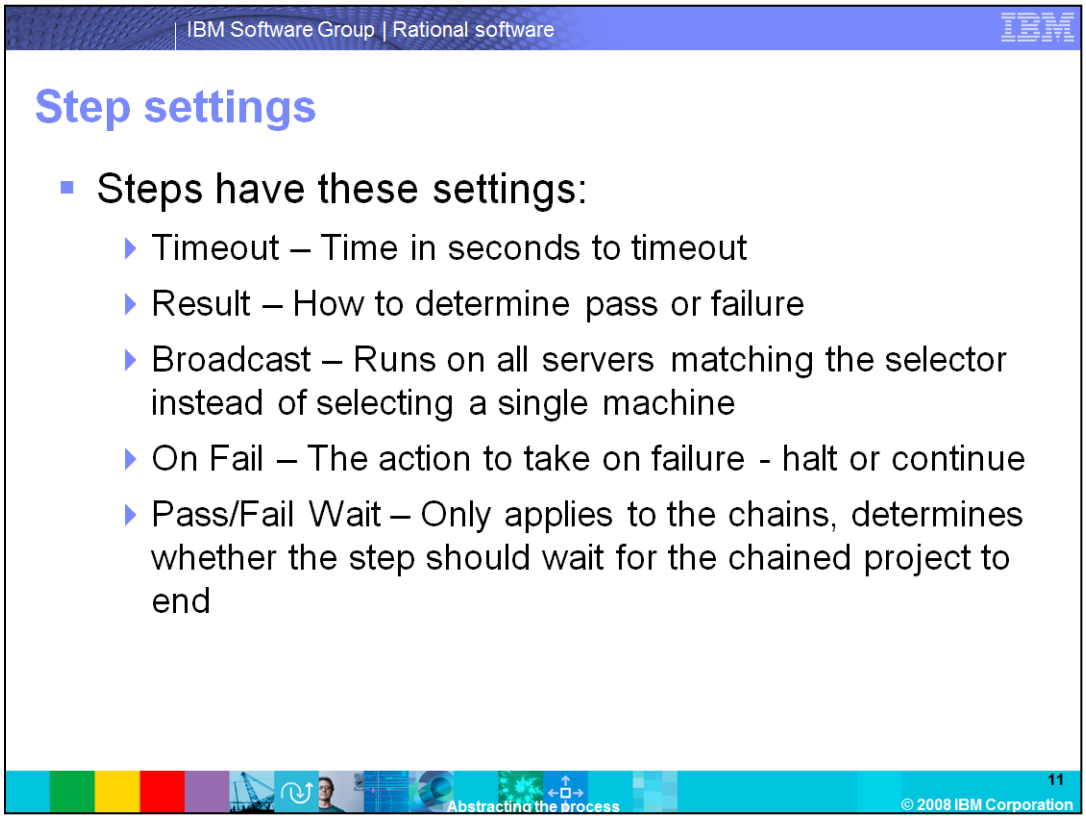
The **Pass/Fail Chain, Start/Pass/Fail Notify, Class, Selector and Environment** are all the same as the project, only now defined for the step. Note that the environment is still cumulative, so there are potential entries from the server, project, and step. Also, the Selector entry overwrites the Project 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:\.



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.

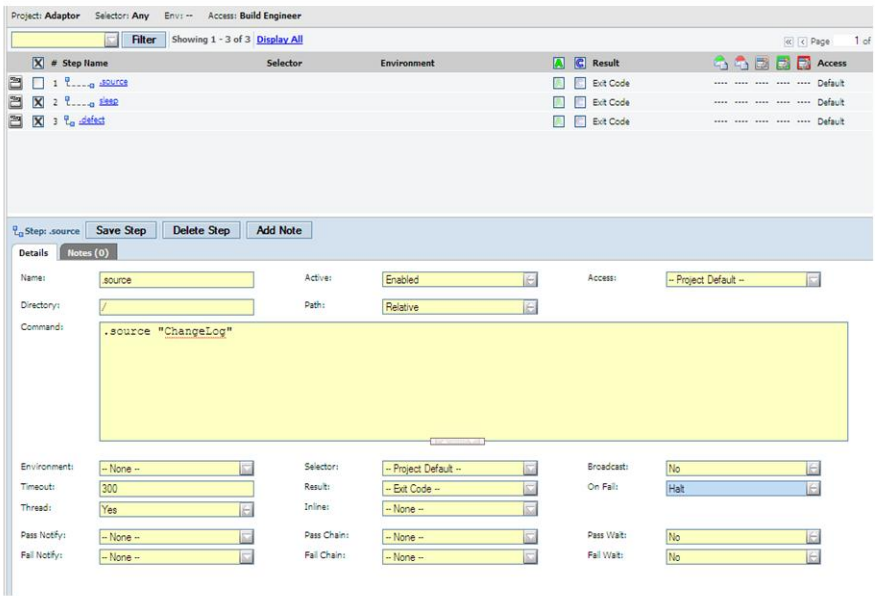
Pass/Fail Wait applies when Pass or Fail chains are defined for the step. It determines that if a Pass or Fail chain has been defined, then the step should wait for that Chain to complete before continuing on to the next step.

Core of the step

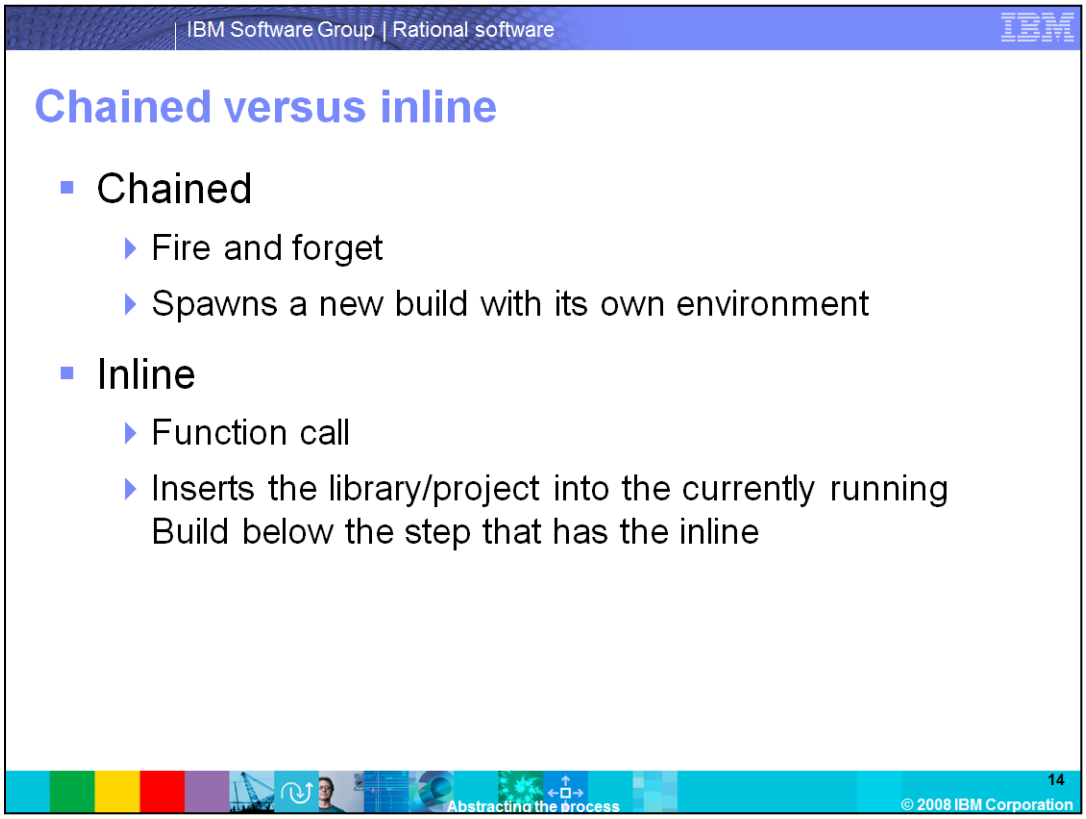
- The last setting is the command field.
- Command field defines a command that runs on the agent on whatever shell is configured for that agent.
- This is the small logical part of the process, and constitutes the building blocks of the Build Forge process.

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



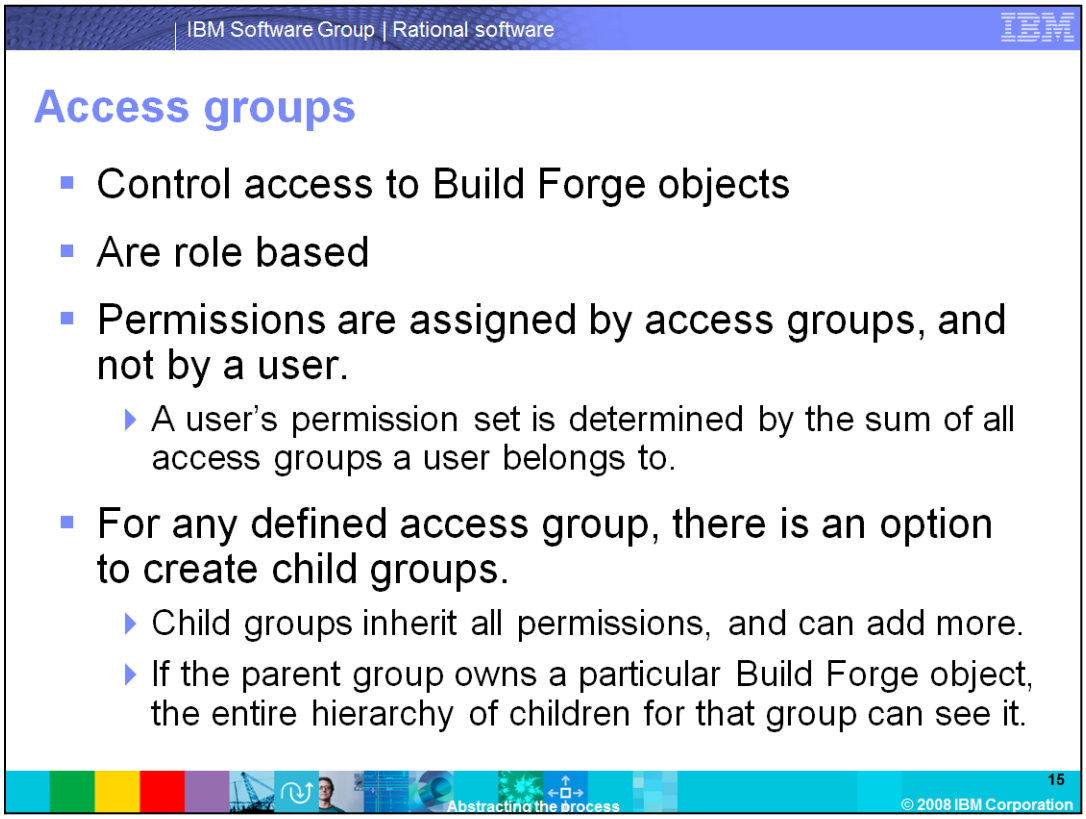
This slide shows the Step menu for a project. Note that in addition to the Pass/Fail notifies, there are also pencil icons. These define Inline, Pass, and Fail Chain, from left to right. Moving the mouse over these icons shows the values of the settings. When moving the mouse over the far left program 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.



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.

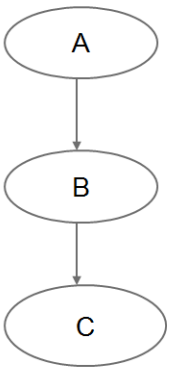
Inline is similar to a function call, as everything still occurs in the currently running Build. If a project or library is inlined for a step, it is as if those steps had been cut out from the project or library and pasted below the step defining the inline. Any changes to the environment are reflected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



Access Groups 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.

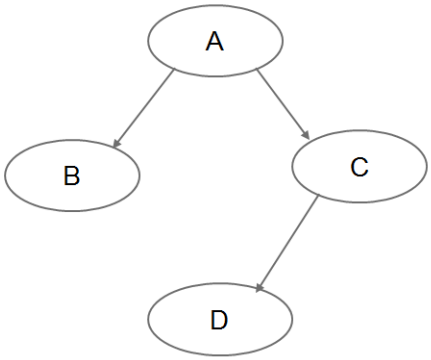
Access group diagram



- 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



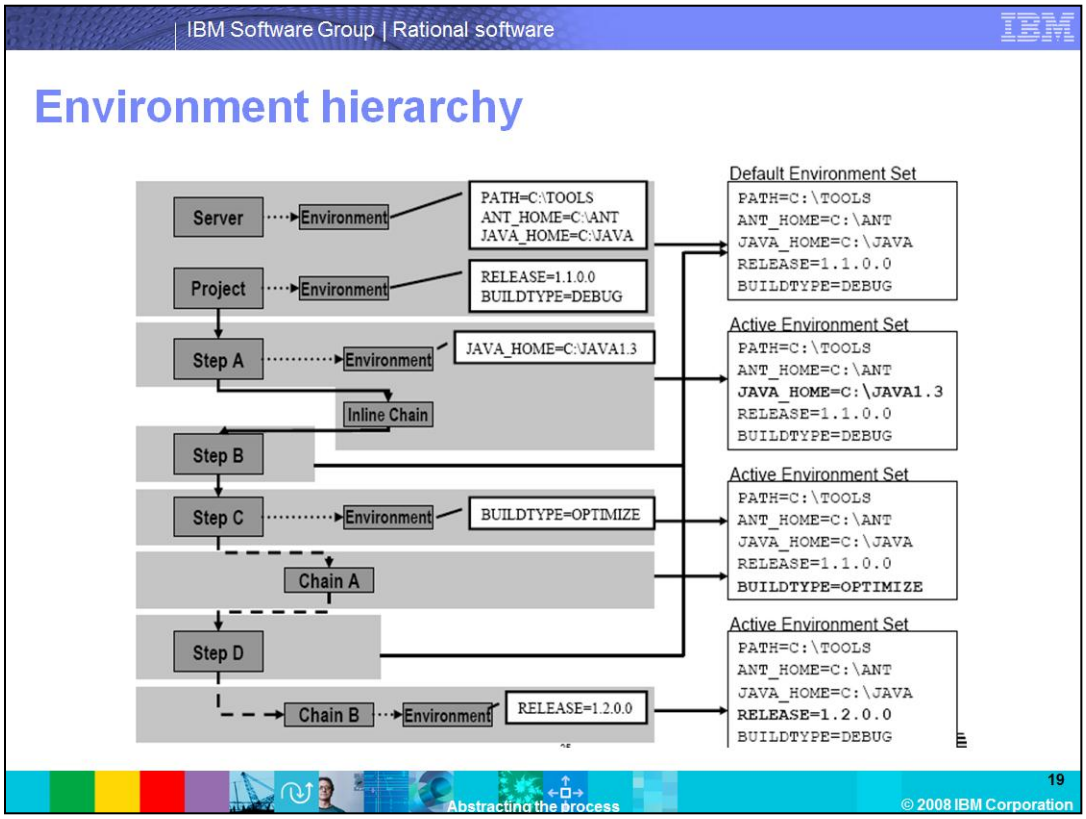
- If a project is owned by B, only B can see that project. As far as groups in A, C and D are concerned, that project does not exist.
- If a user belongs to both group B AND D, that user is able to see any project owned by any group A, B, C, or D.

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.

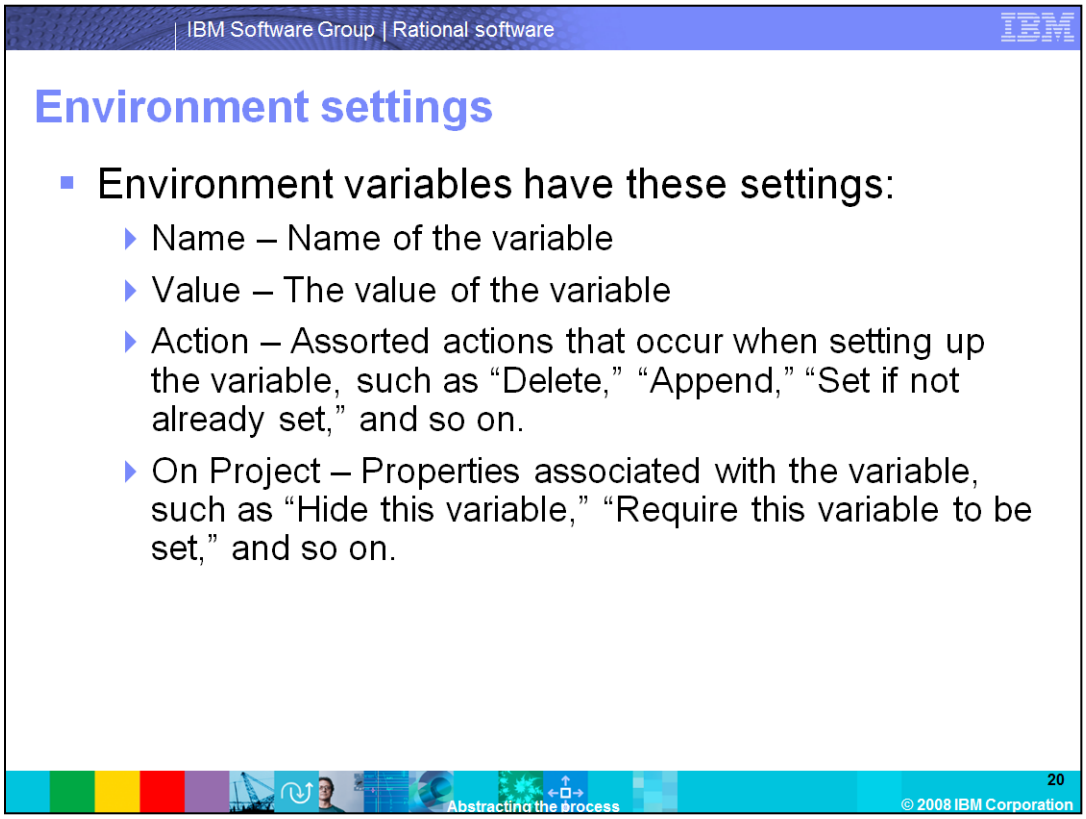
Environments

- 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.



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.



The environment configuration has several settings:

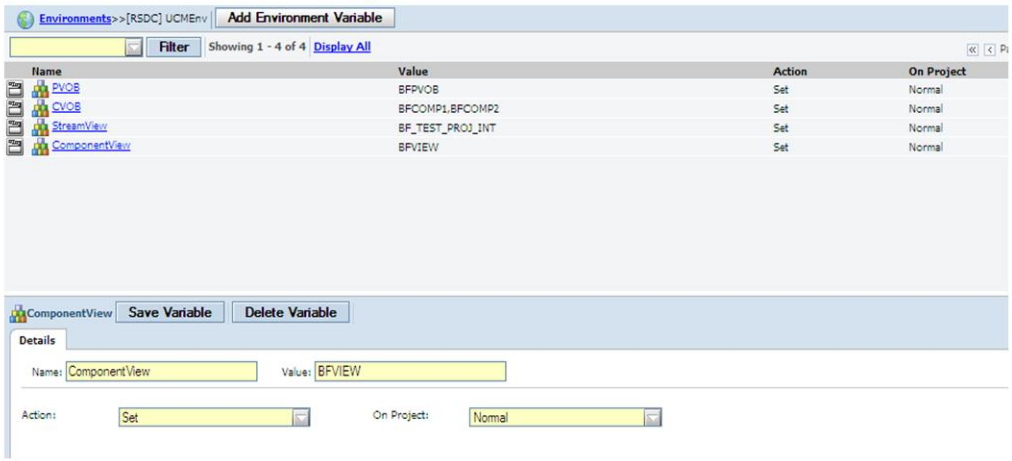
Name declares the variable and sets its value.

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 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.

Environment settings



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.

Trademarks, copyrights, and disclaimers