This module covers the basics of building builds 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

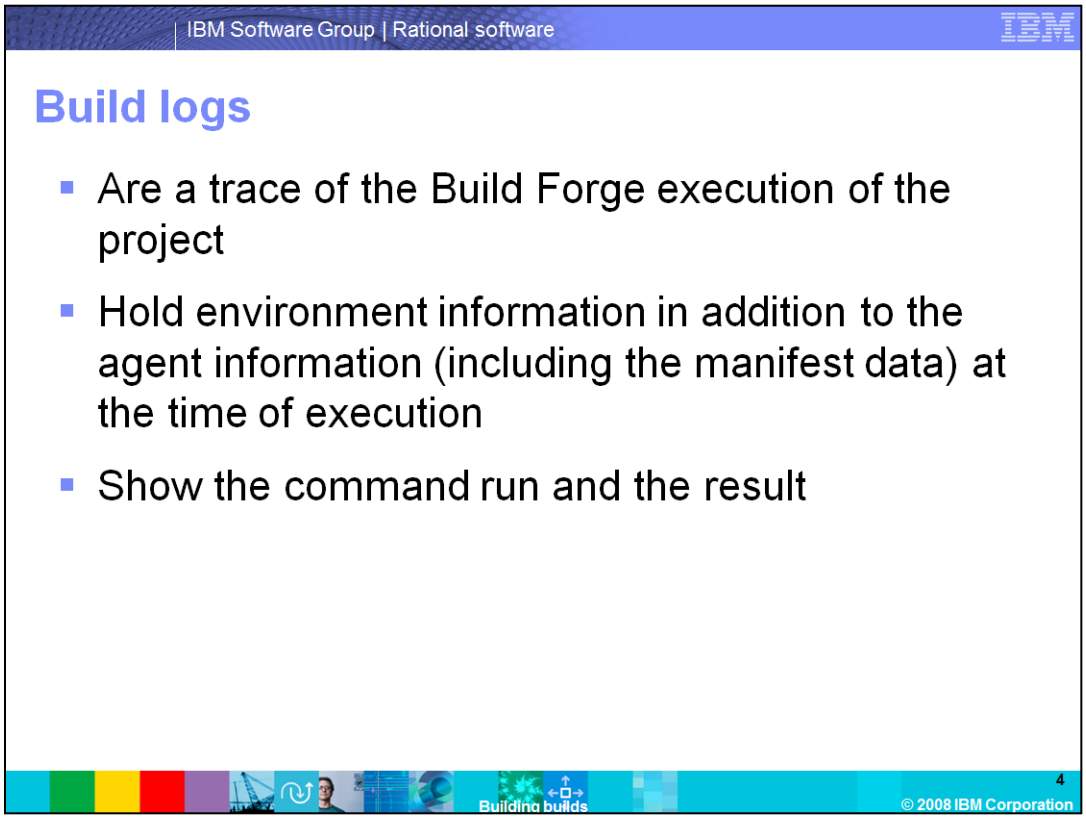
- Know how to read, filter and navigate a Build Forge build log
- Understand what classes and purges are
- Be able to schedule builds and purges
- Know what the bill of materials is

This module discusses Build Forge builds, and the parts associated with the build. It further details the Build Forge build log and how to read, filter and navigate it. The bill of materials is covered as well. Classes are explored, and this module describes what they are used for and how they are related to purges. Finally, this module explains scheduling.

What is a build?

- When a project is run, the result is captured in the build.
- Build logs exist on the database, but the physical files are kept in the agent directory.

When a project is run, the build is the result. The build is a collection of logs compiled by the result of each of the steps in the project. The logs are kept on the console database, but the build artifacts are kept in the agent where they were generated by default.



The build log captures everything that occurs during the execution of the project. Generally, this shows the manifest of the Server where the step ran, along with the environment information. It also shows the execution and the results of that step.

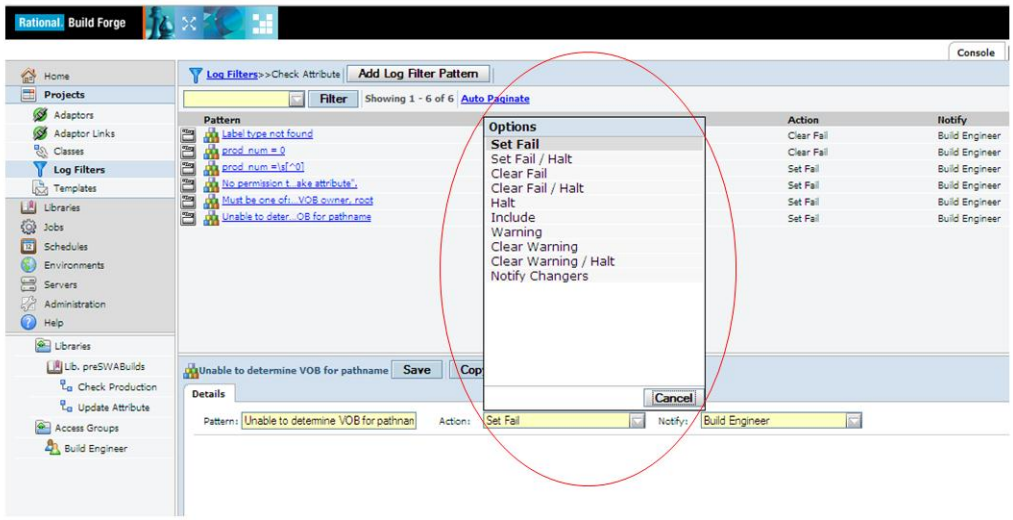
Log filters

- The default action taken for all steps is to go off the exit code of the command. However, this is not always accurate (see: Ant scripts).
- Filters define a collection of one or many regular expressions to determine a trigger and an action to take when it is tripped
- Filters allow for greater freedom in determining pass, warning, and failure of steps during a Build
  - This is important when determining branching logic for the step.

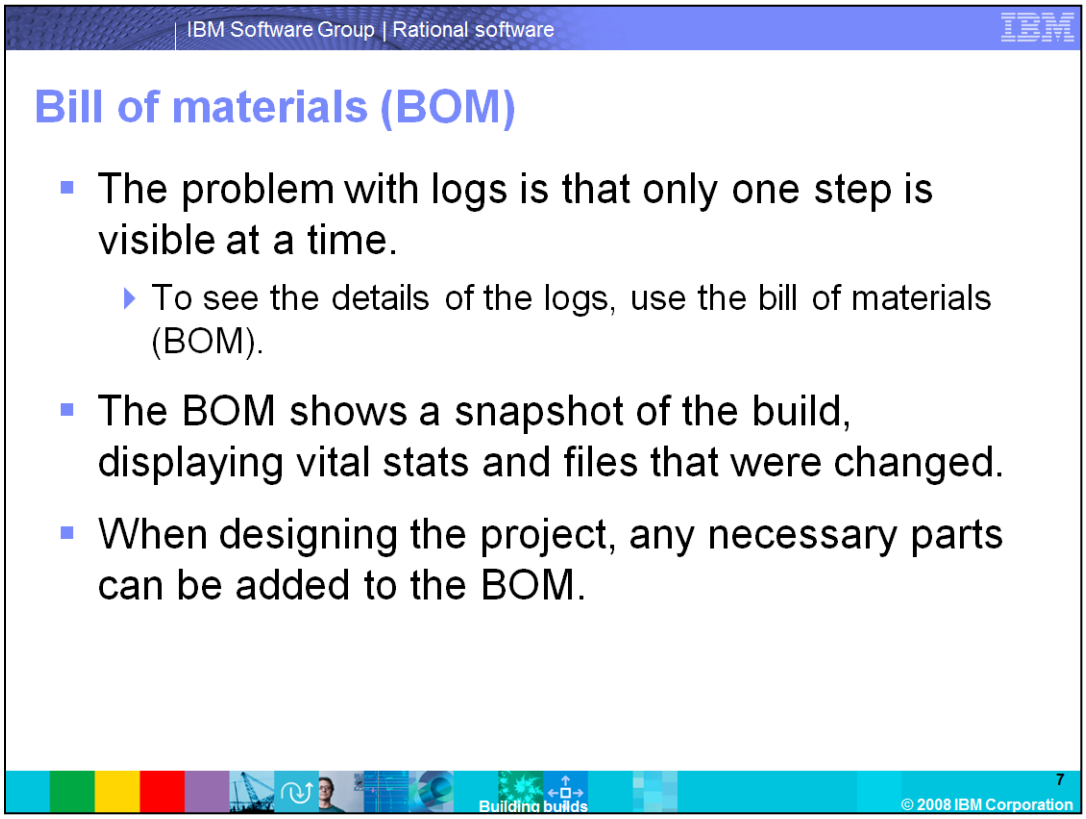
A prevailing problem is how to determine if a step was a pass or a fail. By default, the step looks at the exit code of the command that was run. If the exit code was zero, then the step passes. However, any non-zero value fails the step. This potentially can be used to base a step pass or failure on. One example where this does not work is with Ant scripts. Ant scripts always return an exit code of one, even when they are successful.

The Build Forge solution for this situation is to define a log filter. The log filter is a collection of regular expressions that define actions to take when the regular expression is tripped. This is particularly powerful, as multiple actions can be defined that will resolve from top to bottom in the step log. For example, a particular regular expression can be set to fail the build, but if another condition occurs, it can be changed to pass. This allows for very flexible rules in determining a step's final state.

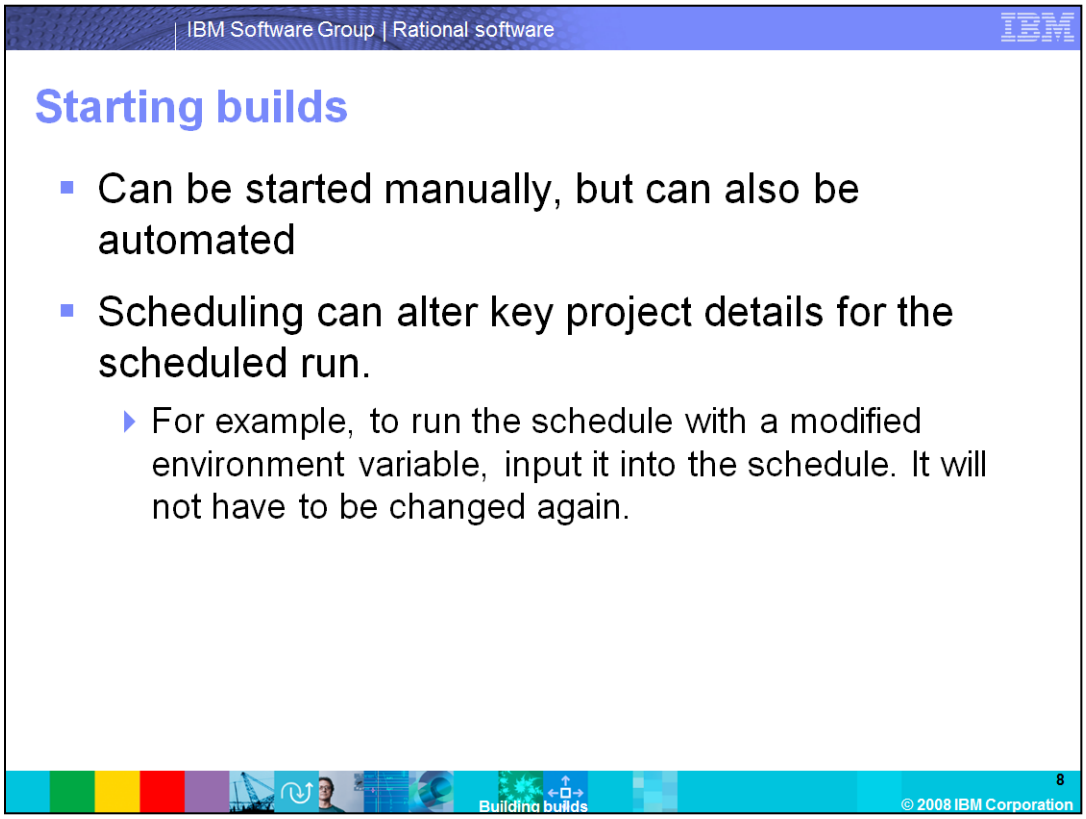
Log filter detail



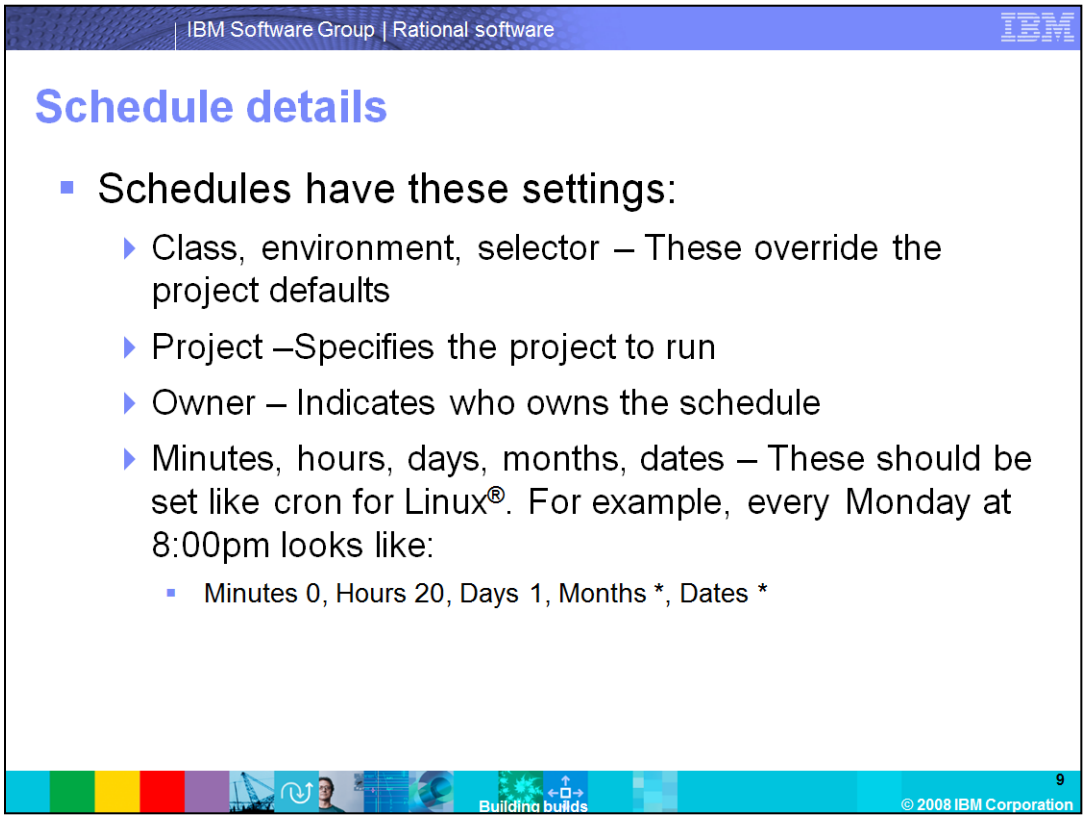
This slide shows the screen for setting up a log filter. Note the available options for usable actions when a particular regular expression is tripped. The expressions are evaluated top to bottom, so multiple rule log filters should be set accordingly.



The steps in the build produce the logging information, but that can be unwieldy, especially when projects can scale to the magnitude of several hundred steps. In such a situation, it becomes a hassle to click through each step to determine what happened in the build. To avoid this, use the bill of Materials (BOM). The BOM is the Build Forge method for condensing data into something easily and quickly perused. While designing the project, particular information can be designated for the BOM. When the build is complete, users can open the BOM and quickly get a sense of what happened during the build.



Build Forge offers scheduling capabilities for setting up specific Build runs. This allows for the automation of the execution of Builds in the Management Console. When a Build is scheduled, the console allows users to set up options specific to that Build. In particular, a user can redefine the Selectors, Environment variables, and other options for that Build.



The Schedule has several configuration options:

Class, Environment, and Selector can all be overridden for the schedule. Additionally, the Schedule allows users to specifically change the value of the variables in the environment for the scheduled Build.

Project defines the project the schedule is being set for.

Owner sets up who owns the Schedule. Currently, the Schedule is owned by a user and not by an access group, which is different from all other ownership dynamics in the Management Console.

Minutes, Hours, Days, Months, and Dates all define the schedule. The format is the same as cron for Linux.

What to keep

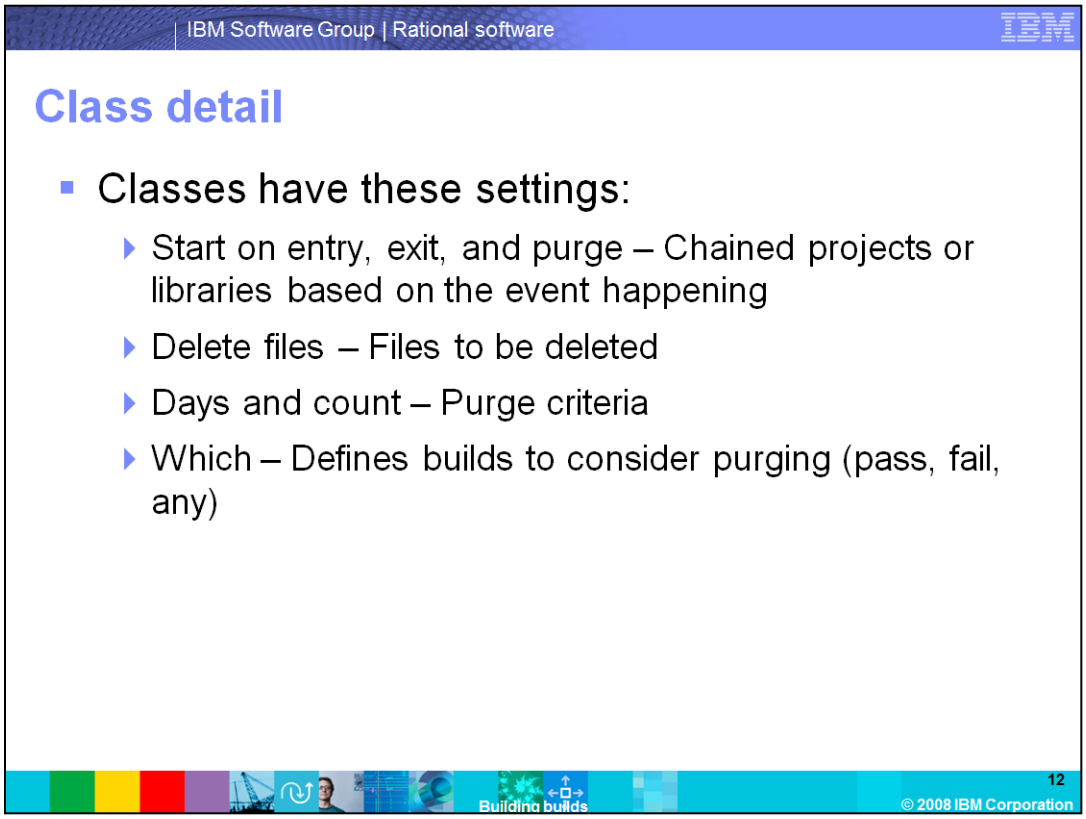
- Use purges to keep necessary builds and delete the unnecessary ones.
  - Purges remove builds and files from the Build Forge system.
  - Note: Purges do not show up on the console as running, but do consume agent job slots.

Purges are the delete function for Build Forge. Purges delete the Build log, files, and whatever other information might be stored on the Build. Note that purges consume job slots on the agent, as they must go to the agent to clean up files that might have been left behind. The purge does not show up in the Build queue though. This is further discussed later in this module.

Classes

- Builds can be manually selected for deletion. This is a process automation tool though, so Build Forge can do the work instead.
- Classes define what criteria to use when checking a build to see if it should be purged.
  - For example, to keep the last three builds, set the class criteria to keep three builds. The next refresh cycle will delete the rest automatically.

Classes are used in Build Forge for automatically determining what Builds to purge. Classes allow users to define what criteria to consider when deciding whether a Build should be purged from the console or not. The two criteria that can be set up are Build number and age. For example, a class can be established to delete all Builds of a particular project older than three days, or the Builds exceeding age five.



Classes have varied configuration information:

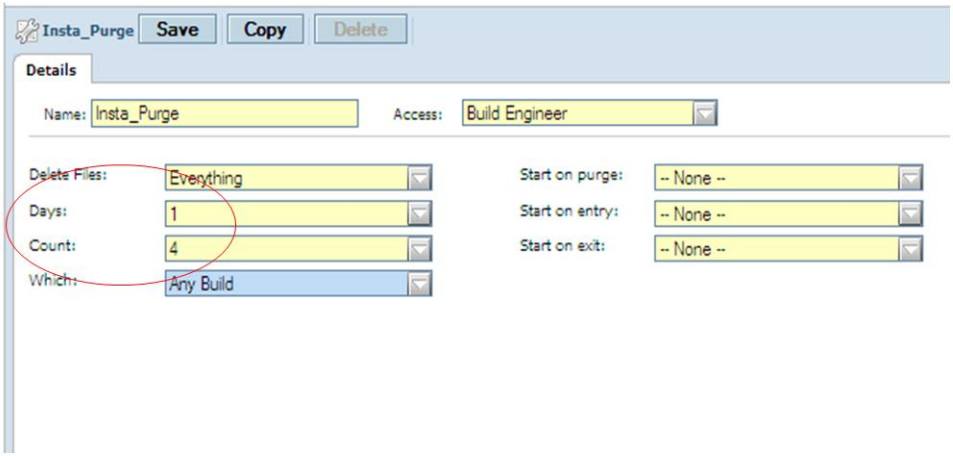
Start of Entry, Exit, and Purge set up chained actions to take on particular events. Entry and Exit are events defined as when a Build is Entering or Exiting this particular class. For example, if a user has additional actions to take before moving the Build from Scratch to Production, users can set those up here. The Purge event happens upon purging.

Delete File defines what the class should delete when the purge takes place. There is a mix of options to delete just the data on the console, just the Build artifacts on the Agents, or a combination of both.

Days and Count set up the criteria for determining what Builds should be purged.

Which defines the Builds to consider with the Days and Count criteria, whether it should be all Builds, or just failed Builds.

Class detail

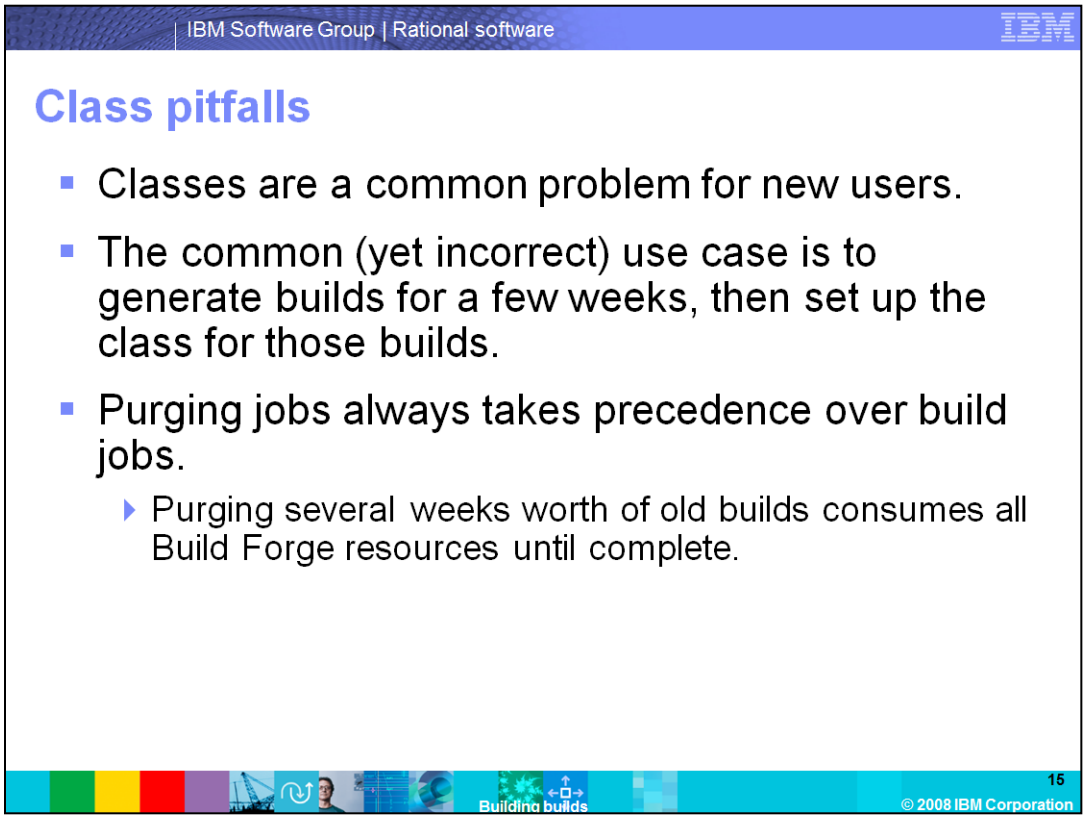


This slide shows the configuration menu for the Class. Note here that the Days and Count are pre-determined, and are not completely definable.

Schedules revisited

- By default, purges are checked periodically by the engine.
- The class purge check can be scheduled like a project.
- Project is a field for the schedule:
  - Class purge schedule can be selected in that field.
  - When selected, the class field becomes the target class whose purge cycle is scheduled.

By default, the Build Forge console checks every 15 minutes to see if there are any new builds eligible for purging based on their class. However, this can become too cumbersome if several builds appear for purge simultaneously. Upon rechecking the schedule, the project setting on the schedule can alternatively be set to class purge schedule. If that is set, the class field in the schedule becomes the class purge cycle being defined. This allows a particular time to be set that the console will consider a class for purging. Doing this avoids the problem of having too many builds appear for purge at one time.



The class mechanism is a common problem for new users. This stems from the Build Forge default class, whose production does not delete any builds. If a user has been using a project for a few weeks and has accumulated a stack of builds, then decides to set up the class to something else more practical, those builds eligible for purge will begin purging simultaneously. This becomes a performance problem, as the purge job takes precedence over regular build jobs in the engine. Since the purge does not appear on the console job queue, it shows the engine running very hard, but not doing anything. To avoid this situation, ensure that the classes are set up before starting any builds to ensure that the build count stays under control.

Summary

- Builds are instantiations of projects.
- Builds are a collection of logs from the execution of the steps defined by the project.
- Schedules automate the execution of builds.
- Classes define the criteria for deleting builds no longer needed.
- Always set up classes before starting builds.

In summary, builds are the collection of logs from running a project. They show all the actions that happened, and what the results of those actions were. A schedule can be set for both builds and purges in Build Forge. Finally, to avoid trouble with excess builds in the future, always set up a class for the project to be run on.

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.

Linux is a registered trademark of Linus Torvalds 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 NONINFRINGEMENT. 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.

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.