Leveraging Rational Team Concert's build capabilities for Continuous Integration

Krishna Kishore
Senior Engineer, RTC
IBM
Krishna.kishore@in.ibm.com

IBM Software
Innovate2011
The Premier Event for Software and Systems Innovation

Software. Everyware.

August 9-11, Bangalore | **August 11,** Delhi

Agenda

- ➢What Is Continuous Integration
- ➢Overview of Rational Team Concert Components
- ➢Overview of Jazz Build System
  - ❖Jazz Build Engine
  - ❖Jazz Build Definition
  - ❖Jazz Build Process
- ➢Continuous Integration Key Practices with Rational Team Concert
- ➢Demo
- ➢Questions and Answers
  - ❖Please feel free to ask questions during the presentation

What is Continuous Integration

Continuous integration is a software development best practice that distributed teams use more and more as a way to mitigate integration problems and to facilitate development of cohesive software more rapidly.

Continuous Integration Key Practices

The effort required to integrate a system increases exponentially with time. Continuous integration is a software development practice that promotes frequent team integrations and automatic builds. By integrating the system more frequently, integration issues are identified earlier, when they are easier to fix, and the overall integration effort is reduced. The result is a higher-quality product and more predictable delivery schedules.

(referencing Martin Fowler "Continuous Integration" white paper)

Benefits of Continuous Integration

➢**Improved feedback**: Continuous integration shows constant and demonstrable progress.

➢**Improved bug detection**: Continuous integration enables you to detect and remove errors early, often minutes after they've been injected into the product.

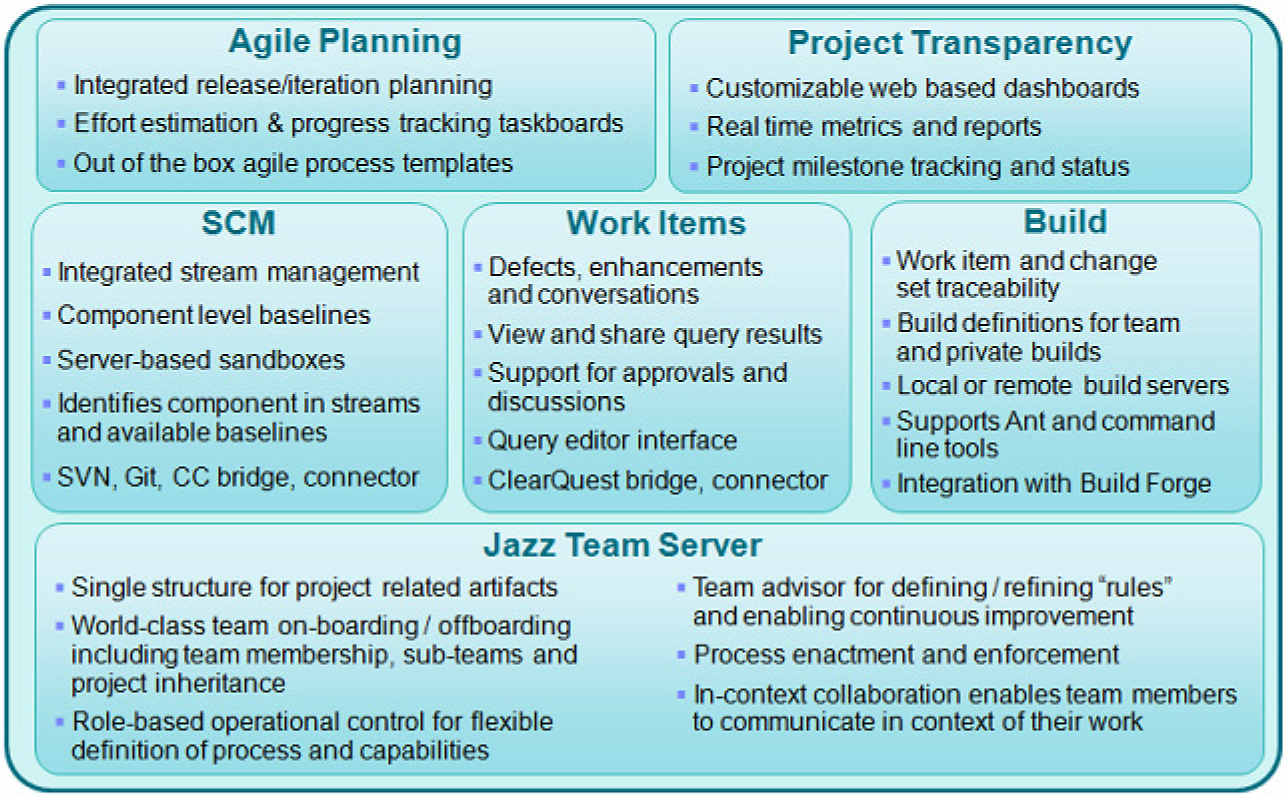
➢**Improved collaboration**: Continuous integration enables team members to work together safely. They know that they can make a change to their code, integrate the system, and determine very quickly whether or not their change conflicts with others.

➢**Improved system integration**: By integrating continuously throughout your product, you know that you can actually build the system, thereby mitigating integration surprises at the end of the lifecycle.

Benefits of Continuous Integration - Continued

- **Reduced** number of parallel changes that need to be merged and tested.
- **Reduced** number of errors found during system testing: All conflicts are resolved before making new change sets available and by the person who is in the best position to resolve them.
- **Reduced** technical risk: You always have an up-to-date system to test against.
- **Reduced** management risk: By continuously integrating your system, you know exactly how much functionality that you have built to date, thereby improving your ability to predict when and if you are actually going to be able to deliver the necessary functionality.

Rational Team Concert Components



Agile Planning
- Integrated release/iteration planning
- Effort estimation & progress tracking taskboards
- Out of the box agile process templates

Project Transparency
- Customizable web based dashboards
- Real time metrics and reports
- Project milestone tracking and status

SCM
- Integrated stream management
- Component level baselines
- Server-based sandboxes
- Identifies component in streams and available baselines
- SVN, Git, CC bridge, connector

Work Items
- Defects, enhancements and conversations
- View and share query results
- Support for approvals and discussions
- Query editor interface
- ClearQuest bridge, connector

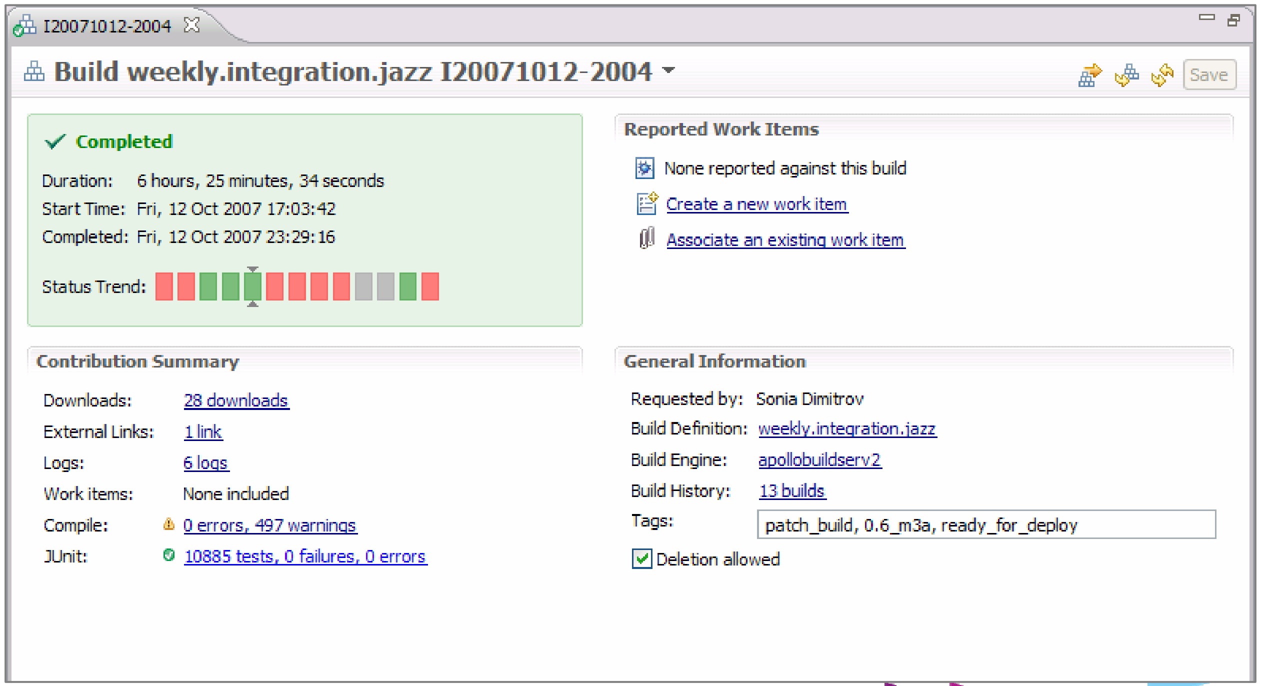
Build
- Work item and change set traceability
- Build definitions for team and private builds
- Local or remote build servers
- Supports Ant and command line tools
- Integration with Build Forge

Jazz Team Server
- Single structure for project related artifacts
- World-class team on-boarding / offboarding including team membership, sub-teams and project inheritance
- Role-based operational control for flexible definition of process and capabilities
- Team advisor for defining / refining "rules" and enabling continuous improvement
- Process enactment and enforcement
- In-context collaboration enables team members to communicate in context of their work

Overview of Jazz Build System

Jazz Build System

- **A Build is a first-class object**
  - Associated build results, tests or any artifact
  - Full navigability to all artifacts contributing to a build
  - Automatically schedule and maintain history of builds
  - Supports personal builds
- **Build engine is "pluggable"**
  - Cruise Control
  - Build Forge
- **Build terms**
  - Build definition
  - Build engine
  - Build Requests



Build weekly.integration.jazz I20071012-2004

✓ **Completed**

Duration:	6 hours, 25 minutes, 34 seconds
Start Time:	Fri, 12 Oct 2007 17:03:42
Completed:	Fri, 12 Oct 2007 23:29:16
Status Trend:	🟥🟥🟩🟩🟩🟥🟥🟥🟥⬜⬜🟩🟥

Reported Work Items

None reported against this build
Create a new work item
Associate an existing work item

Contribution Summary

Downloads:	28 downloads
External Links:	1 link
Logs:	6 logs
Work items:	None included
Compile:	0 errors, 497 warnings
JUnit:	10885 tests, 0 failures, 0 errors

General Information

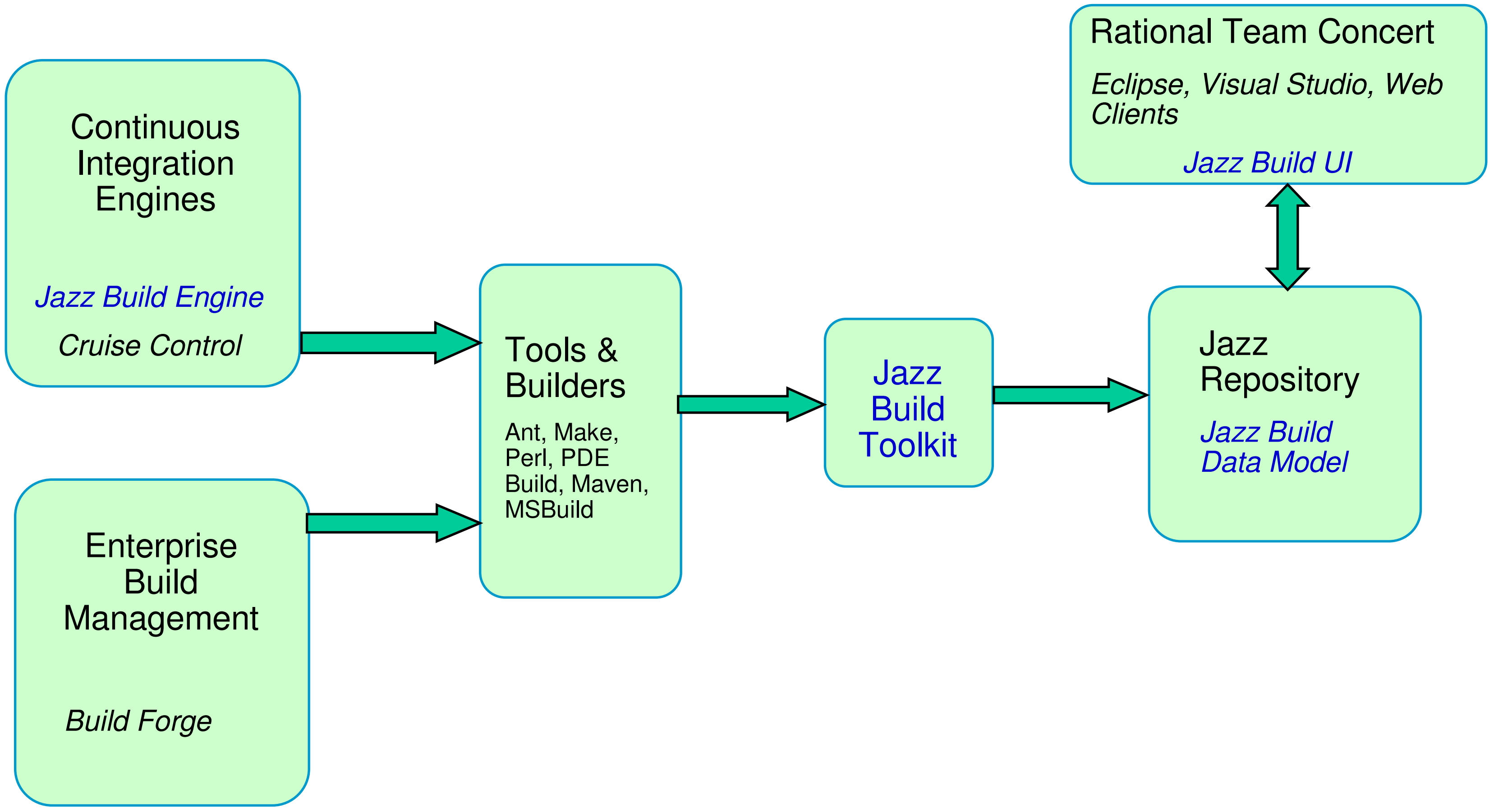
Requested by:	Sonia Dimitrov
Build Definition:	weekly.integration.jazz
Build Engine:	apollobuildserv2
Build History:	13 builds
Tags:	patch_build, 0.6_m3a, ready_for_deploy

[✓] Deletion allowed

Jazz Build System - Continued

- Easy to use wizards help create:
  - Build Engines
  - Build Definitions
    - Scheduled
    - Continuous, at specified intervals
    - On demand by team members
- Changes delivered for builds are easily seen by interested parties.
  - Able to see who requested a build.
  - Easy to compare 2 builds to see the differences between them.
  - Easy to identify who delivered code to a build.
- Can use any method for builds
  - MSBuild, ant, batch files, Perl scripts, Maven, COBOL compilers, etc…
  - Cruise Control, Build Forge, and other Automation tools
  - Very good integration with Ant and the Build Toolkit.

The Jazz Build Environment

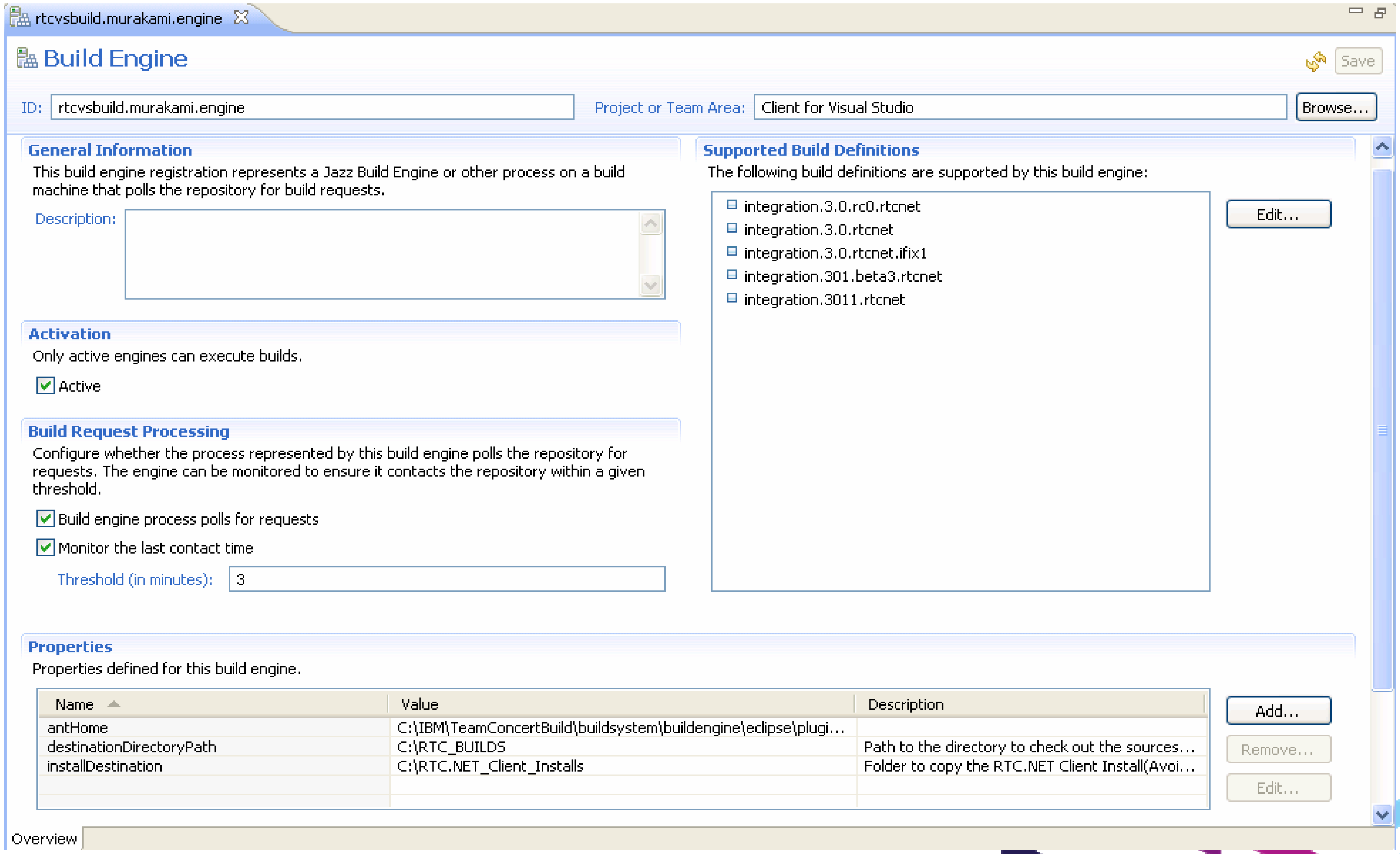


Jazz Build Engine

- Represents a build system running on a build server.
- To begin using Team Concert builds you need to create a "Build Engine". A Build Engine is used by one or more Build Definition(s).
- Easy to use wizards help create Build Engines
  - Most steps are contained in the wizard.
  - There are some steps that have to be done separately.
- Can have Build Engines for each project
  - Able to assign a specific Build Engine to a particular project.
  - If you have a lot of projects, hard to maintain.
- Can use one Build Engine for multiple projects
  - Build Engines are run serially, so you can use the same one to run several builds.

Jazz Build Build Engine - Continued

- Example of the Build Engine Wizard



rtcvsbuild.murakami.engine

Build Engine

Save

ID: rtcvsbuild.murakami.engine

Project or Team Area: Client for Visual Studio

Browse...

General Information

This build engine registration represents a Jazz Build Engine or other process on a build machine that polls the repository for build requests.

Description:

Activation

Only active engines can execute builds.

☑ Active

Build Request Processing

Configure whether the process represented by this build engine polls the repository for requests. The engine can be monitored to ensure it contacts the repository within a given threshold.

☑ Build engine process polls for requests

☑ Monitor the last contact time

Threshold (in minutes): 3

Supported Build Definitions

The following build definitions are supported by this build engine:

- integration.3.0.rc0.rtcnet
- integration.3.0.rtcnet
- integration.3.0.rtcnet.ifix1
- integration.301.beta3.rtcnet
- integration.3011.rtcnet

Edit...

Properties

Properties defined for this build engine.

Name	Value	Description
antHome	C:\IBM\TeamConcertBuild\buildsystem\buildengine\eclipse\plugi...	
destinationDirectoryPath	C:\RTC_BUILDS	Path to the directory to check out the sources...
installDestination	C:\RTC.NET_Client_Installs	Folder to copy the RTC.NET Client Install(Avoi...

Add... Remove... Edit...

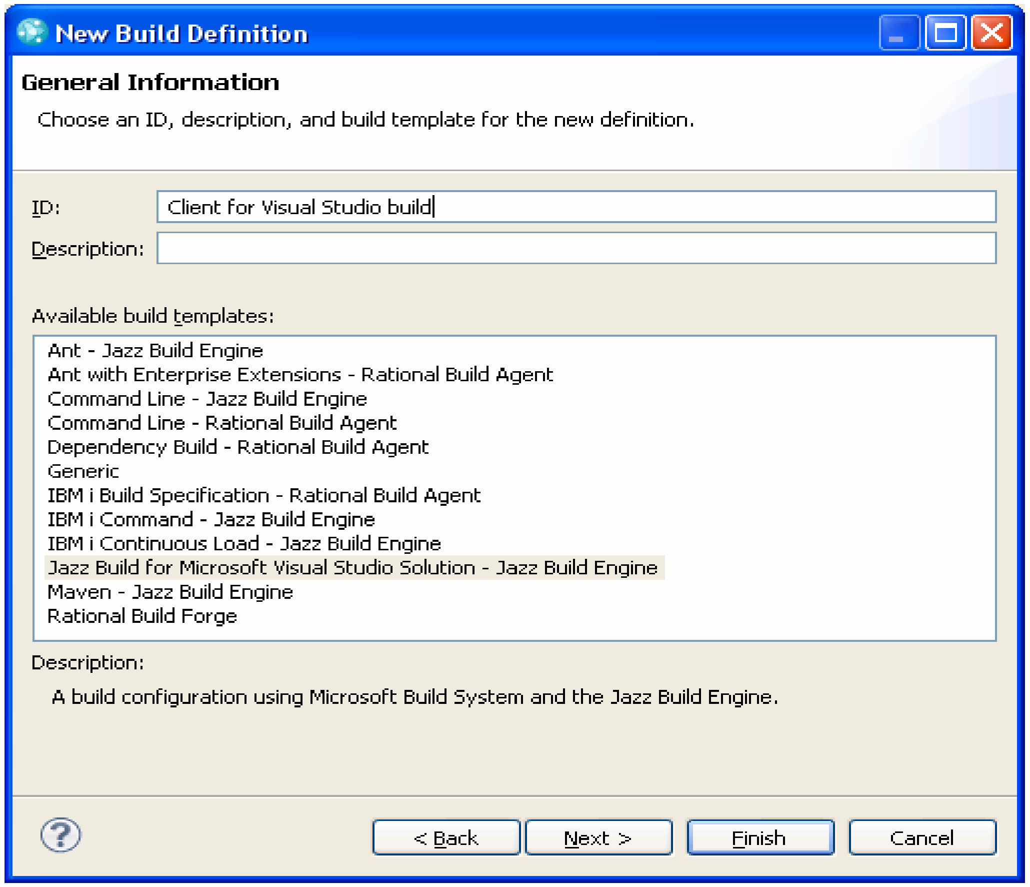
Overview

Jazz Build Definition

- Defines a build, such as a weekly project-wide integration build.
- Easy to use wizards help create Build Definitions
  - Define variables to use
  - Define scripts to use
  - Define command line arguments to use
  - Define Workspace to use and which Components to exclude
- Build Definitions can be several, like for each team.
  - One used by Development team
  - One used by the QA team
  - One used by the Release Team
- Build Definitions can run on single or multiple Build Servers.
- Can create a New Build Definition or copy an existing one.

Jazz Build Definition - Continued

- Provides many out of the box build templates



General Information

Choose an ID, description, and build template for the new definition.

ID: Client for Visual Studio build

Description:

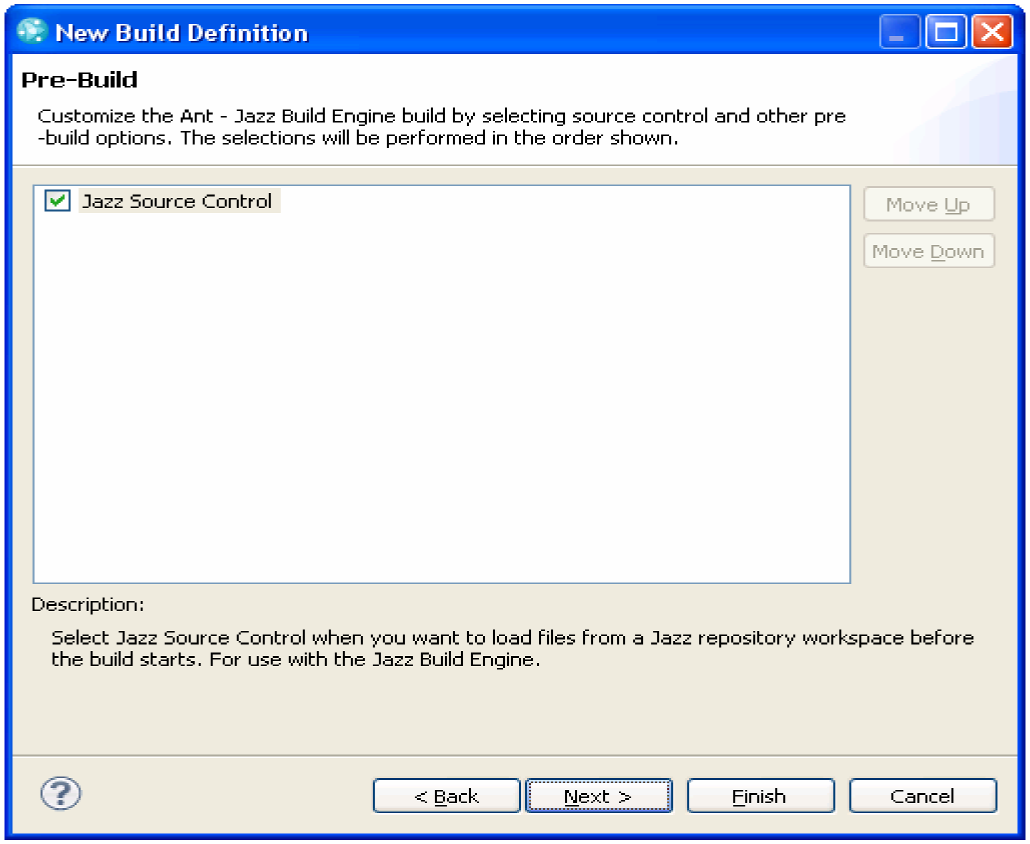
Available build templates:

Ant - Jazz Build Engine
Ant with Enterprise Extensions - Rational Build Agent
Command Line - Jazz Build Engine
Command Line - Rational Build Agent
Dependency Build - Rational Build Agent
Generic
IBM i Build Specification - Rational Build Agent
IBM i Command - Jazz Build Engine
IBM i Continuous Load - Jazz Build Engine
Jazz Build for Microsoft Visual Studio Solution - Jazz Build Engine
Maven - Jazz Build Engine
Rational Build Forge

Description:

A build configuration using Microsoft Build System and the Jazz Build Engine.

Jazz Build Definition – Pre Build step



Pre-Build

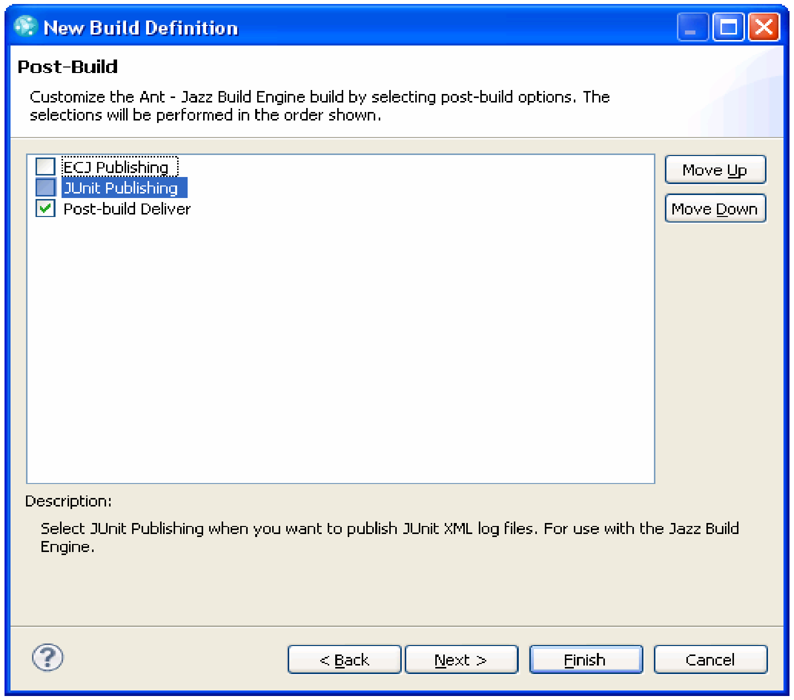
Customize the Ant - Jazz Build Engine build by selecting source control and other pre -build options. The selections will be performed in the order shown.

[✓] Jazz Source Control

Description:

Select Jazz Source Control when you want to load files from a Jazz repository workspace before the build starts. For use with the Jazz Build Engine.

Jazz Build Definition – Post Build step



New Build Definition

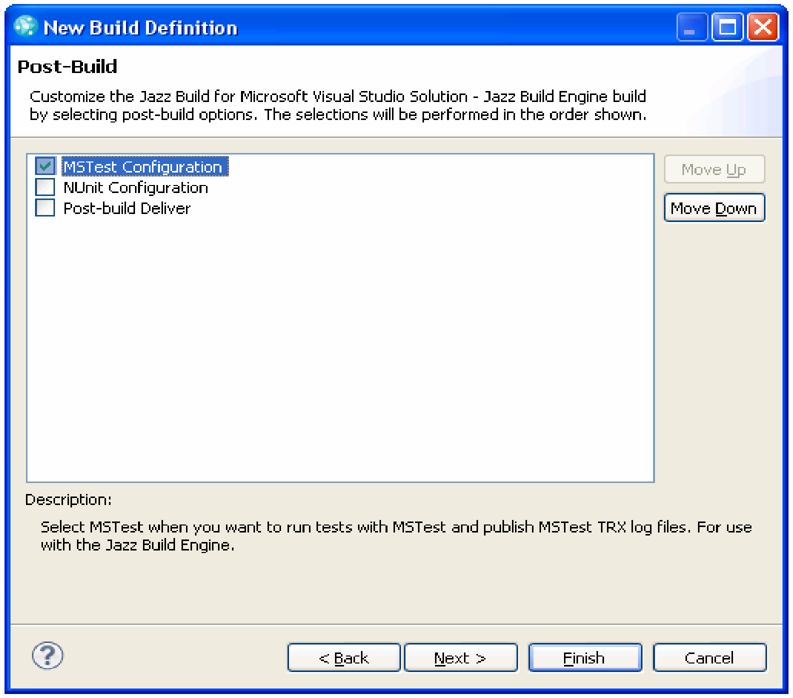
Post-Build

Customize the Ant - Jazz Build Engine build by selecting post-build options. The selections will be performed in the order shown.

- [ ] ECJ Publishing
- [ ] JUnit Publishing
- [✓] Post-build Deliver

Description:

Select JUnit Publishing when you want to publish JUnit XML log files. For use with the Jazz Build Engine.



New Build Definition

Post-Build

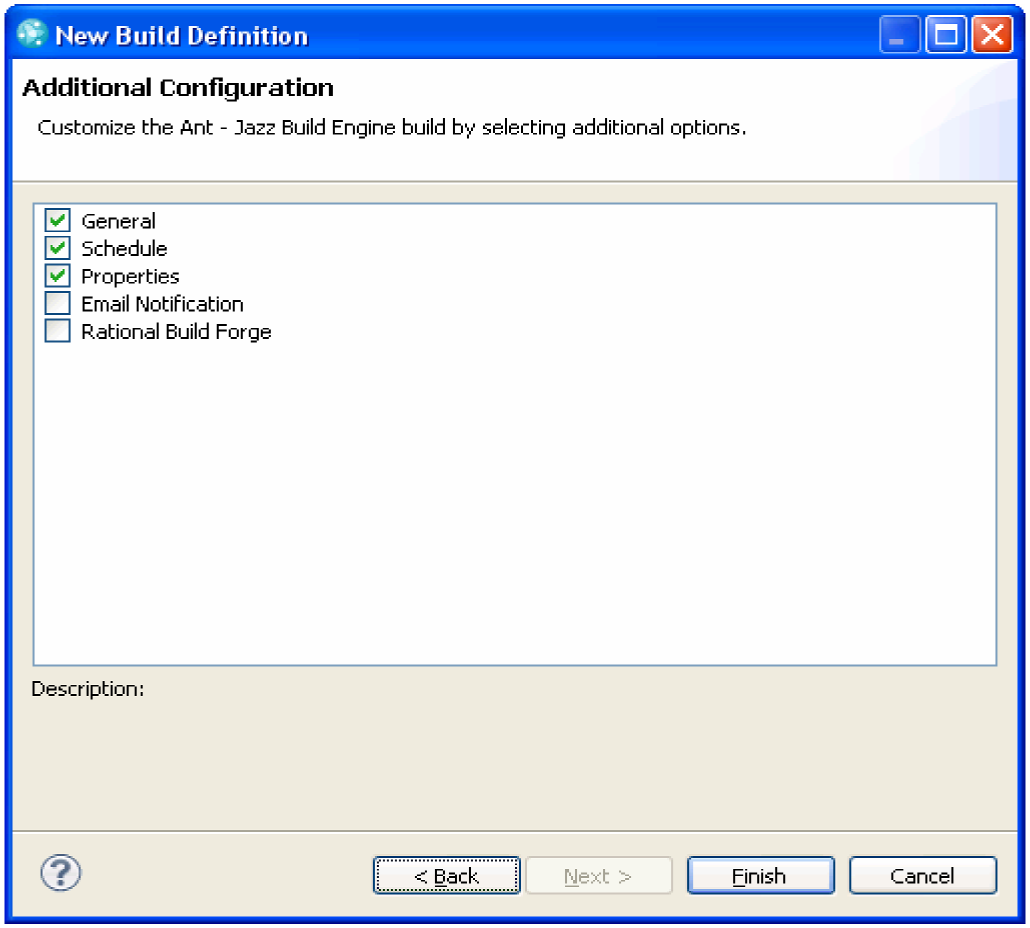
Customize the Jazz Build for Microsoft Visual Studio Solution - Jazz Build Engine build by selecting post-build options. The selections will be performed in the order shown.

- [✓] MSTest Configuration
- [ ] NUnit Configuration
- [ ] Post-build Deliver

Description:

Select MSTest when you want to run tests with MSTest and publish MSTest TRX log files. For use with the Jazz Build Engine.

Jazz Build Definition –Additional Configurations



New Build Definition

Additional Configuration

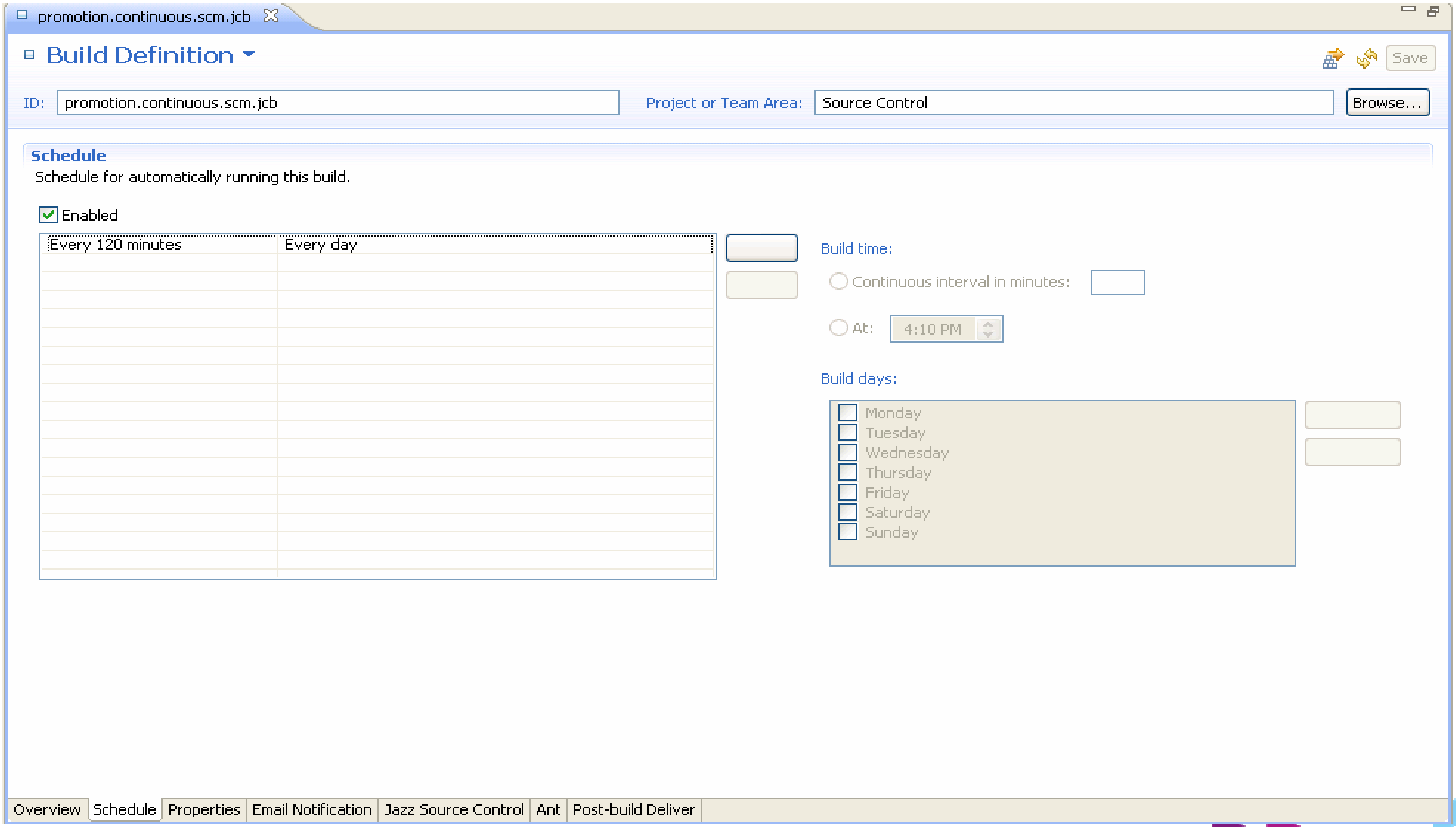
Customize the Ant - Jazz Build Engine build by selecting additional options.

- [✓] General
- [✓] Schedule
- [✓] Properties
- [ ] Email Notification
- [ ] Rational Build Forge

Description:

Jazz Build Process: Build Definition Schedule

- Can make it Continuous or schedule a time for the build to run.



promotion.continuous.scm.jcb

Build Definition

ID: promotion.continuous.scm.jcb

Project or Team Area: Source Control

Schedule

Schedule for automatically running this build.

[✓] Enabled

Every 120 minutes	Every day

Build time:

( ) Continuous interval in minutes:

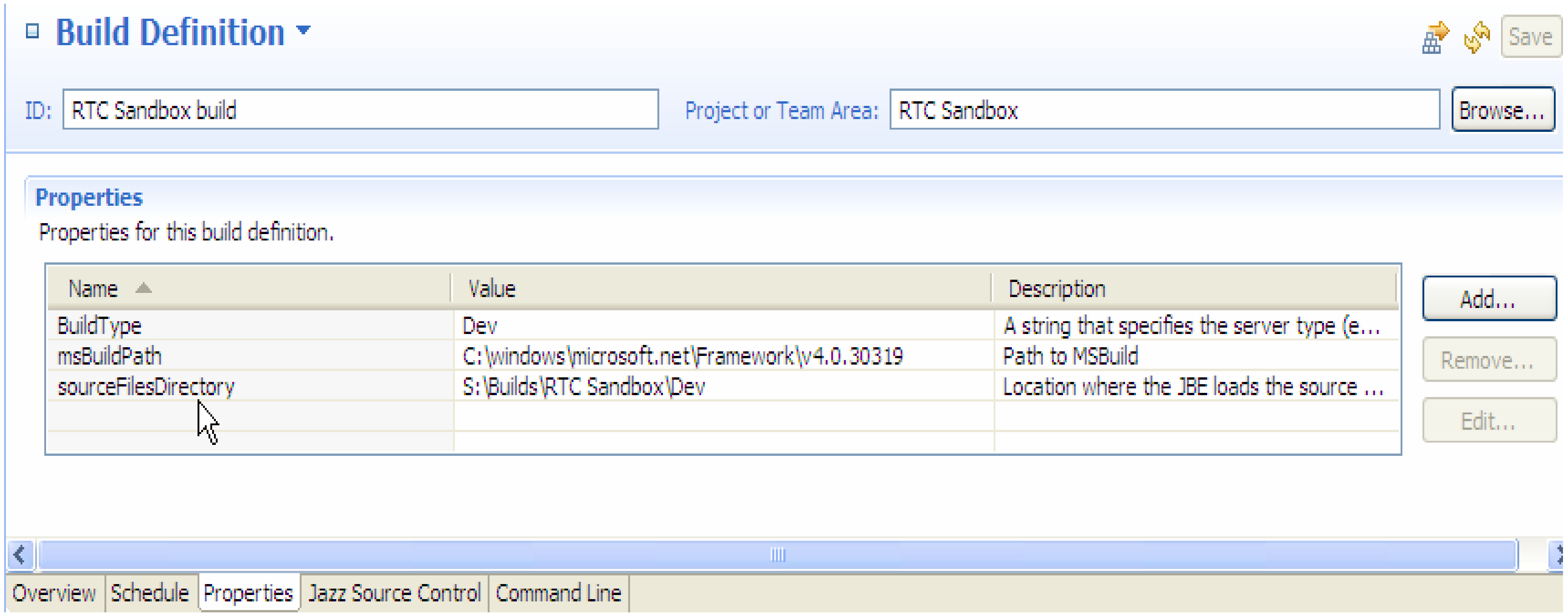
( ) At: 4:10 PM

Build days:

- [ ] Monday
- [ ] Tuesday
- [ ] Wednesday
- [ ] Thursday
- [ ] Friday
- [ ] Saturday
- [ ] Sunday

Jazz Build Process: Build Definition - Properties

- You can add properties that will be used by the build.
- These properties can be used in other sections of the Build Definition.



ID: RTC Sandbox build

Project or Team Area: RTC Sandbox

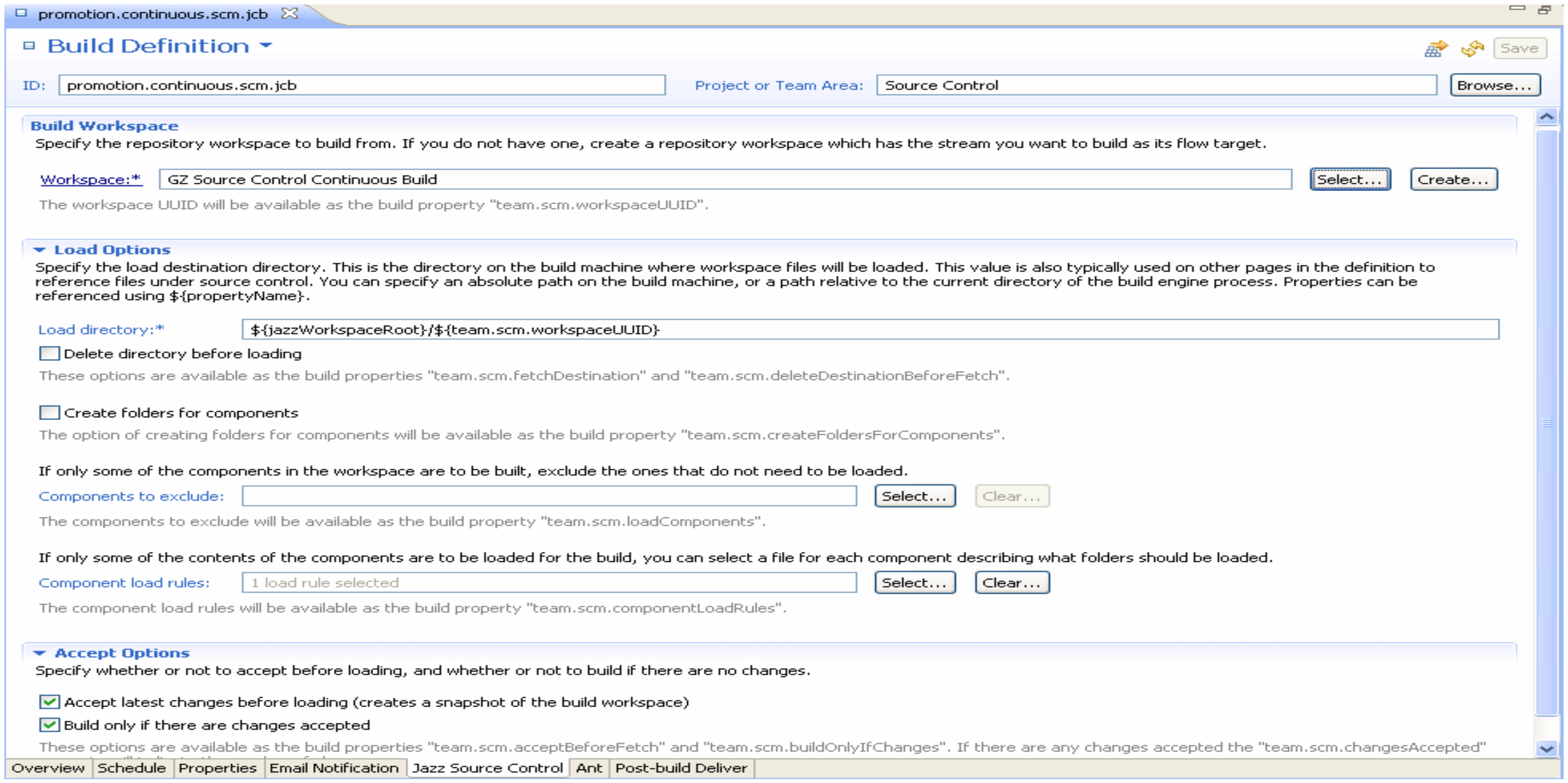
Properties

Properties for this build definition.

Name	Value	Description
BuildType	Dev	A string that specifies the server type (e...
msBuildPath	C:\windows\microsoft.net\Framework\v4.0.30319	Path to MSBuild
sourceFilesDirectory	S:\Builds\RTC Sandbox\Dev	Location where the JBE loads the source ...

Jazz Build Process: Build Definition – Jazz Source Control

- This is where you specify the workspace to use for the build.
- Several other items can be selected here to make the build perform in various ways, as you can see.



promotion.continuous.scm.jcb

Build Definition

ID: promotion.continuous.scm.jcb Project or Team Area: Source Control Browse...

Build Workspace

Specify the repository workspace to build from. If you do not have one, create a repository workspace which has the stream you want to build as its flow target.

Workspace:* GZ Source Control Continuous Build Select... Create...

The workspace UUID will be available as the build property "team.scm.workspaceUUID".

Load Options

Specify the load destination directory. This is the directory on the build machine where workspace files will be loaded. This value is also typically used on other pages in the definition to reference files under source control. You can specify an absolute path on the build machine, or a path relative to the current directory of the build engine process. Properties can be referenced using ${propertyName}.

Load directory:* ${jazzWorkspaceRoot}/${team.scm.workspaceUUID}

☐ Delete directory before loading

These options are available as the build properties "team.scm.fetchDestination" and "team.scm.deleteDestinationBeforeFetch".

☐ Create folders for components

The option of creating folders for components will be available as the build property "team.scm.createFoldersForComponents".

If only some of the components in the workspace are to be built, exclude the ones that do not need to be loaded.

Components to exclude: Select... Clear...

The components to exclude will be available as the build property "team.scm.loadComponents".

If only some of the contents of the components are to be loaded for the build, you can select a file for each component describing what folders should be loaded.

Component load rules: 1 load rule selected Select... Clear...

The component load rules will be available as the build property "team.scm.componentLoadRules".

Accept Options

Specify whether or not to accept before loading, and whether or not to build if there are no changes.

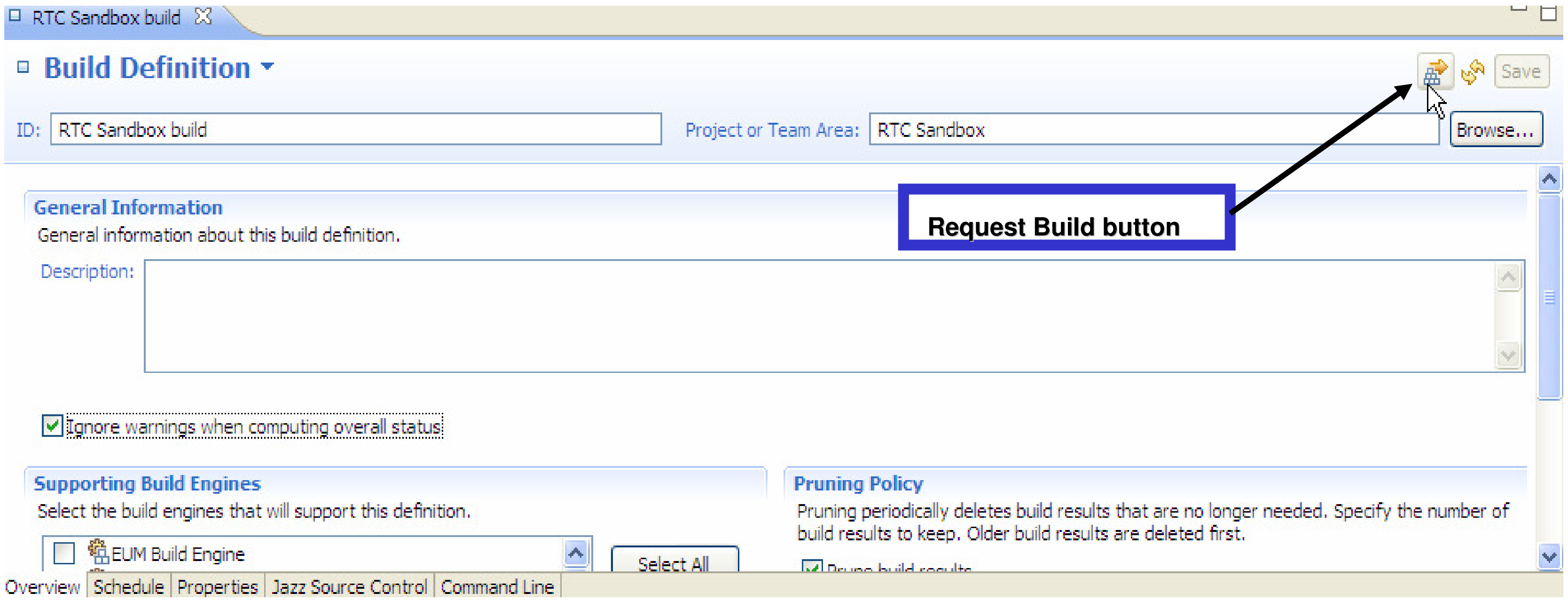
☑ Accept latest changes before loading (creates a snapshot of the build workspace)

☑ Build only if there are changes accepted

These options are available as the build properties "team.scm.acceptBeforeFetch" and "team.scm.buildOnlyIfChanges". If there are any changes accepted the "team.scm.changesAccepted"

Jazz Build Process: Request a Build

- Builds can be run as “Scheduled” or “On Demand”.
- Team members, if permitted, can “request a build”, just by clicking on a button. This can be done in the Eclipse Client and the Visual Studio Client.
- Can request a “Personal Build” or a regular project build.



RTC Sandbox build

Build Definition

ID: RTC Sandbox build

Project or Team Area: RTC Sandbox

General Information

General information about this build definition.

Description:

[✓] Ignore warnings when computing overall status

Supporting Build Engines

Select the build engines that will support this definition.

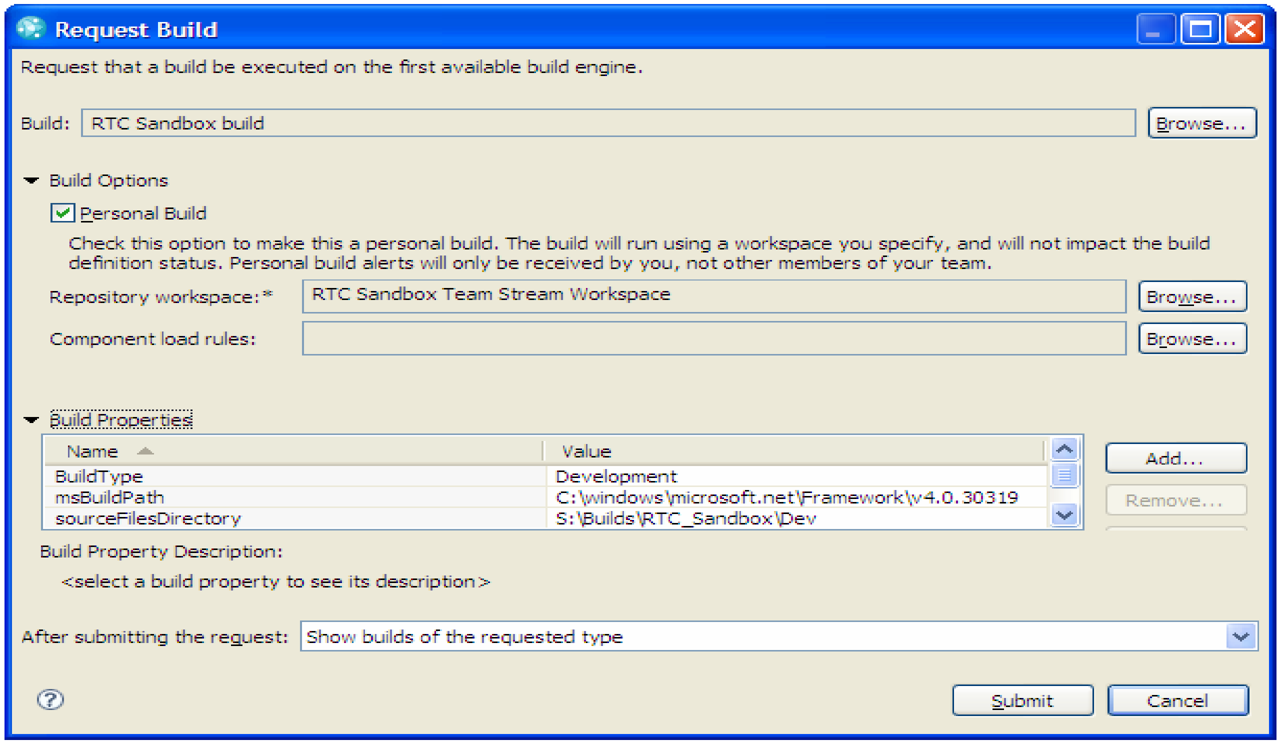
[ ] EUM Build Engine

Pruning Policy

Pruning periodically deletes build results that are no longer needed. Specify the number of build results to keep. Older build results are deleted first.

Jazz Build Process: Personal Build

- Personal builds will run on the build server using the code in the requestor’s workspace.
- Properties of a “Personal Build” can be changed by the requestor. The changes will not affect the project build.



Request that a build be executed on the first available build engine.

Build: RTC Sandbox build

[✓] Personal Build

Check this option to make this a personal build. The build will run using a workspace you specify, and will not impact the build definition status. Personal build alerts will only be received by you, not other members of your team.

Repository workspace:* RTC Sandbox Team Stream Workspace

Component load rules:

Name	Value
BuildType	Development
msBuildPath	C:\windows\microsoft.net\Framework\v4.0.30319
sourceFilesDirectory	S:\Builds\RTC_Sandbox\Dev

Build Property Description:

<select a build property to see its description>

After submitting the request: Show builds of the requested type

Continuous Integration Key Practices with Rational Team Concert

Maintain a Single Source Repository

The base of a Continuous Integration system is to implement a good source control management system to keep track and control all of the files needed to build a product. In this source control repository you must include everything you need for the build.

RTC implements a Source Control component which manages the source code, documents, and other artifacts that a team creates. It provides change-flow management to facilitate sharing of controlled artifacts, retains a history of changes made to these artifacts, and enables simultaneous development of multiple versions of shared artifacts, so that teams can work on several development lines at the same time.

Automate the Build

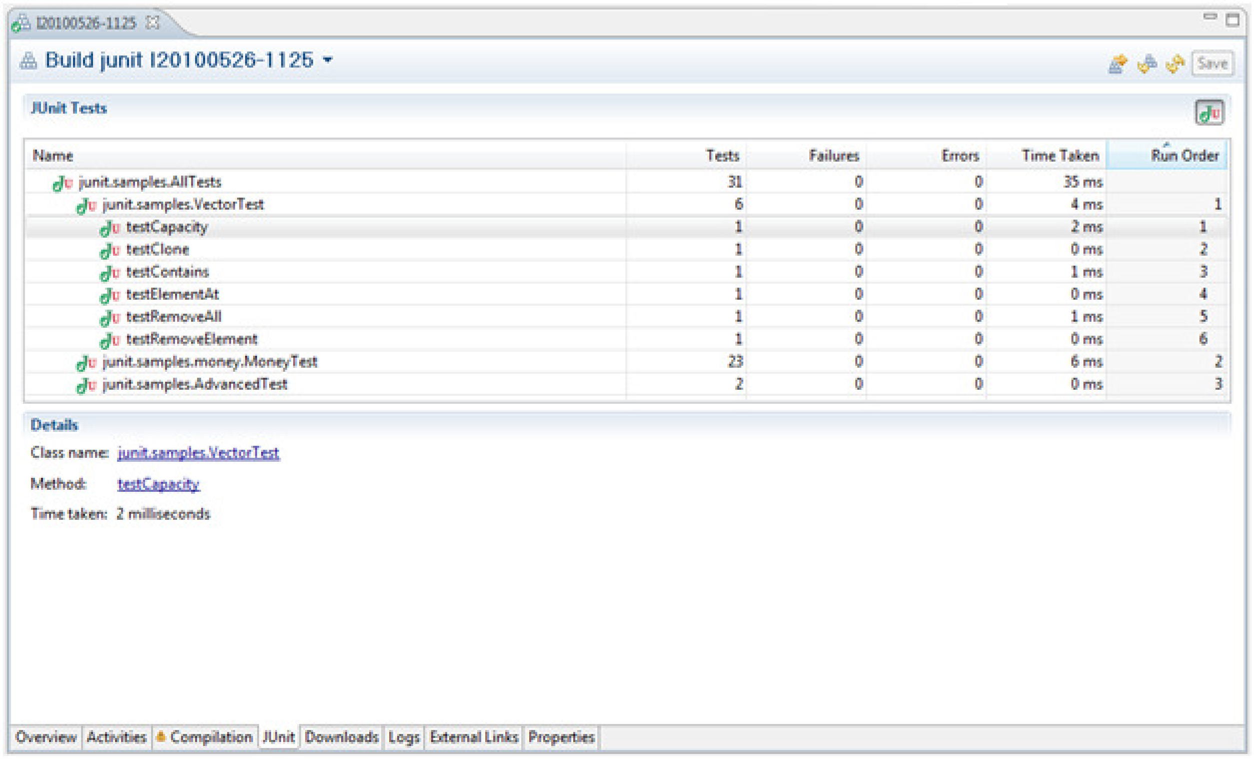
To get an efficient Continuous Integration system you need to implement an automatic build process.

RTC implements a Team Build component which provides support for the automation, monitoring, and awareness of a team's regular builds. This component provides a model for representing the team's build definitions, build engines, and build results. The model supports teams with different build technologies.

Make Your Build Self-Testing

A good practice that will help you to detect errors quickly is to include some automated tests in your build process. If some testing fails, the build should also fail.

RTC is not an IDE, so it doesn't provide features to define and implement unit tests. But in its Team Build component it includes support for execution and result analysis of different unit test frameworks like: CPPUnit, JUnit, MSTest and NUnit. So development teams will be able to execute and check the results of all the unit tests of the project.



Name	Tests	Failures	Errors	Time Taken	Run Order
junit.samples.AllTests	31	0	0	35 ms	
junit.samples.VectorTest	6	0	0	4 ms	1
testCapacity	1	0	0	2 ms	1
testClone	1	0	0	0 ms	2
testContains	1	0	0	1 ms	3
testElementAt	1	0	0	0 ms	4
testRemoveAll	1	0	0	1 ms	5
testRemoveElement	1	0	0	0 ms	6
junit.samples.money.MoneyTest	23	0	0	6 ms	2
junit.samples.AdvancedTest	2	0	0	0 ms	3

Details

Class name: junit.samples.VectorTest

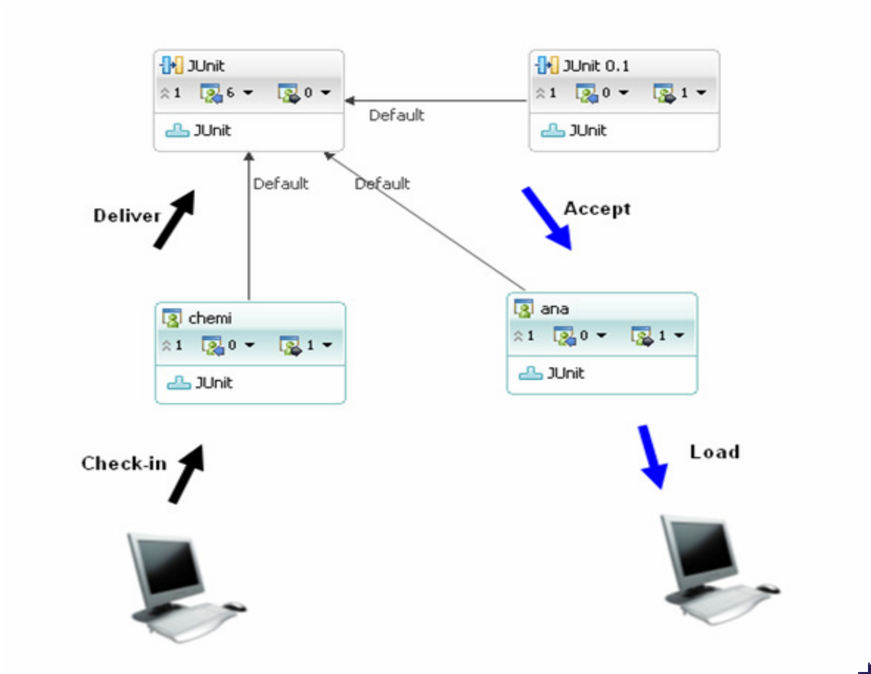
Method: testCapacity

Time taken: 2 milliseconds

Everyone Commits To the Mainline Every Day

This is more a methodology aspect than a technical issue to be resolved or supported by the tool. Communication is key for continuous integration, and developers communicate with others delivering changes they have made to their files. A good practice is to force developers to commit their changes to the main development stream at least once every day.

The RTC Source Control component lets the developer to commit or deliver code to streams as many times as they require as needed . So a development team adopting Continuous Integration practices should encourage its developers to commit at least once per day.



Every Commit Should Build the Mainline on an Integration Machine

Although developers must run local builds and tests in their local machines before delivering to the main development stream, there may be differences between each developer's machine, or code integration errors. This is why it is very important to ensure that an integration build is run on an integration machine each time each a developer commits some changes.

Two ways to achieve this

- developer manually request a build execution after he has committed some new code to the stream
- automatic build will be executed after a code commit to the stream.

RTC supports both the approaches

Keep the Build Fast

It is important to reduce to a minimum the time spent running the integration build each time a developer commits changes to the main development stream.

Test in a Clone of the Production Environment

The test environment as similar as possible to your final production environment.

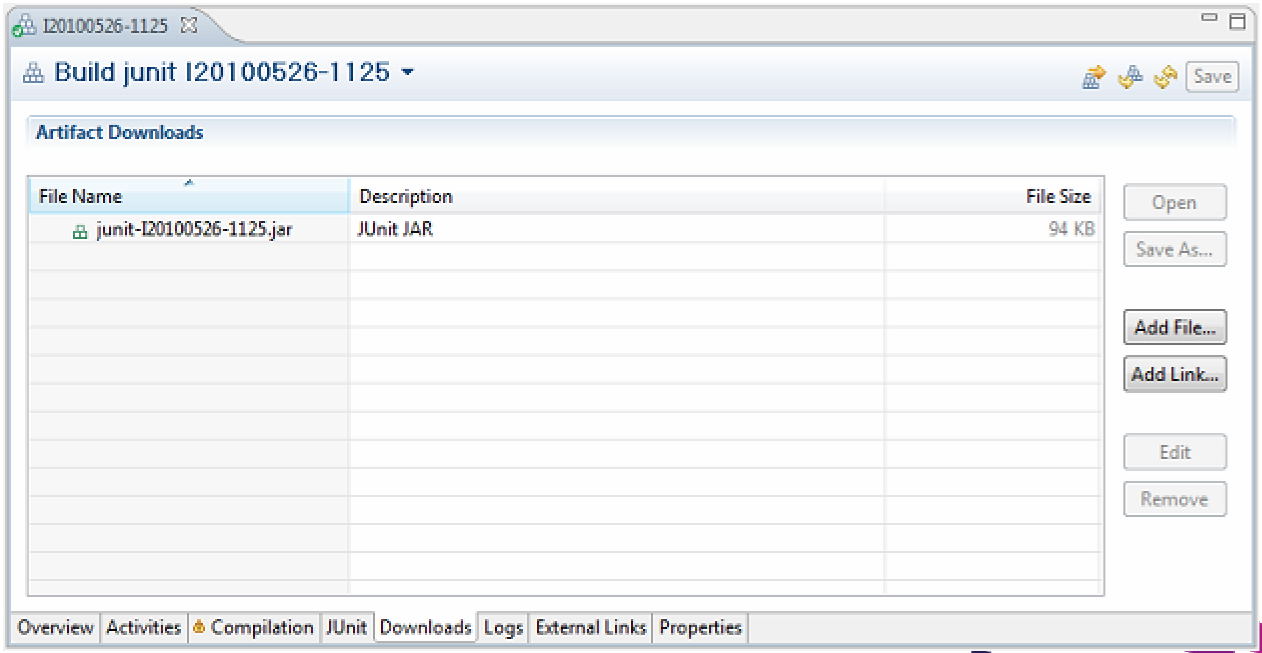
This is more a methodology aspect than a specific feature in RTC.

However, it is worth noting that the RTC Build Team component can be installed multiple times in multiple different environments (all installations working against the same RTC server). Therefore it is very easy to setup several build environments (development, integration and production) executing the builds with different scheduling in each one.

Make it Easy for Anyone to Get the Latest Executable

Most builds produce useful output, such as an executable program, a packaged zip file, or other artifacts. Many people may need to get access to the latest executable to be able to run it or just to see what changed last week. Many times developers are not able to find it because there is not a well known place where these files are stored and they spend many hours just looking for this information.

The Ant build toolkit of RTC has a set of Ant tasks that can perform various operations on a build. Some of these tasks (*artifactLinkPublisher* and *artifactFilePublisher* tasks) enable your build to publish these artifacts. When using these tasks, the artifacts specified in your builds are available in the build result editor, on the Downloads page, once the build finishes.



File Name	Description	File Size
junit-I20100526-1125.jar	JUnit JAR	94 KB

Everyone can see what's happening

Communication is critical to implement a good Continuous Integration system. Each team member needs to have easy and transparent access to the state of the system, last changes made and state of the mainline integration build. Visibility and transparency of the flow of information between team members are essential.

RTC Provides different ways to communicate and track Team Builds

- Build Dashboard viewlets
- Event Notification System for Builds
- History of Builds
- Build Reports
- Build Auditing

Automate Deployment

To implement a Continuous Integration system you need to implement multiple environments (development, integration, production) to run your build and tests and, if possible, automate as much deployment between these environments as possible because you may be doing these deployments several times each day.

www.ibm.com/software/rational

www.ibm.com/software/rational

www.ibm.com/software/rational