



IBM Software Group

Continuous integration using Rational Team Concert

Peter Steinfeld

November 4, 2009



Rational software



Overview

- The importance of using continuous integration
- How to use Rational Team Concert (Jazz) to implement continuous integration for your project
 - ▶ Stream setup
 - ▶ Build setup
 - ▶ Foolproof delivery to the integration stream
 - ▶ How to debug broken builds
- *The conversation you want to have*



The importance of continuous integration

"Happy families are all alike; every unhappy family is unhappy in its own way."

Leo Tolstoy (the first line of Anna Karenina)

- Integration is hard
 - It requires project-wide communication and coordination
 - It frequently uncovers unanticipated problems
- Integration takes time
 - It's difficult to predict how long integration will take
 - Since integration is project wide, a problem in one area can block progress in another
- Integration typically happens at the end of a project, the worst time to introduce risk and delay



Critical aspects of continuous integration

Successful projects practice continuous integration

- ✓ The project has a single main stream of development – the integration stream
- ✓ The integration stream builds successfully every day
- ✓ Automated tests run as part of the build
- ✓ Clear project status is visible via the build results



Perspectives of project stakeholders

Project manager

- ✓ *Knows project status by just looking at the build results.*
- ✓ *Knows that a feature has been implemented when it's in a build.*

Developer

- ✓ *Delivers code and tests to the integration stream continuously.*
- ✓ *Is careful to not break the build.*
- ✓ *Fixes broken builds, no matter who broke it.*

Tester

- ✓ *Build status is clear from the published build results*
- ✓ *Latest good build is the one to test.*



Barriers to continuous integration

- Integration delays of delivery of functionality. It takes extra work for developers to keep pace with the integration stream.
- Integration incurs the overhead of dealing with build problems. Such problems may be unrelated to a specific delivery.
- Integration requires expertise in the associated technologies – configuration management, builds, etc.
- Integration forces developers to communicate with each other and with the build team. Factors such as geographic disbursement can make this a real problem.
- There's never a good time implement continuous integration! We know that it's a good idea, but we're busy writing and debugging code.



Why you need continuous integration

- ✓ Continuous integration reduces surprises and risk.
 - ✓ There are many small integrations throughout the project rather than a few big ones.
 - ✓ There's no "death march" at the end of the project when integration is forced.
- ✓ Continuous integration promotes communication and cooperation among developers.
- ✓ Continuous integration makes project status transparent.
- ✓ Schedules are more predictable.
- ✓ Continuous integration is good for morale. Everyone sees the project making steady progress.



Essential aspects of continuous integration

- ✓ A single integration stream
- ✓ Automated, daily build
 - ✓ Produce the customer deliverable every day
 - ✓ Build runs automated tests – if the tests fail, the build fails
 - ✓ Fix broken builds immediately
- ✓ Transparency
 - ✓ Build success is transparent
 - ✓ Functionality in the build is transparent

"Successful daily builds are the heartbeat of a software project. If you do not have successful daily builds, then you have no heartbeat, and your project is dead!"

Jim McCarthy, Microsoft VC++ product manager



How to set up continuous integration in Rational Team Concert

- **Create an integration stream** – This is where all of the project's source code lives.
- **Each developer creates a personal Jazz workspace that “flows with” the integration stream.** Each developer works in her own Jazz workspace and integrates it regularly with the integration stream.
- **Set up a build of the integration stream.** The build will produce the customer deliverables, run tests, and provide status.



The integration stream

Everyone is collectively responsible for the integrity of the integration stream.

- A Jazz stream captures a set of versions of files
 - ▶ Similar to the concept of a branch in other CM systems
 - ▶ The stream changes when people deliver new content
- Streams contain Jazz components
- Jazz components contain directories and files



A sample Jazz integration stream

The screenshot shows the 'Stream' configuration window in Rational software. It includes the following sections and callouts:

- Stream name:** A callout points to the 'Name' field, which contains 'Build Framework 3.0 - Foundation Stable'.
- Stream description:** A callout points to the 'Description' text area, which contains the text: 'This is the 3.0 work aimed at reworking the jazz based build infrastructure to be more modular, faster, and easier to consume by upstream teams.'
- List of components in the stream:** A callout points to the 'Components' list, which includes items such as 'Dashboard (917: 1001_RC3_20090911b) (Dashboard Viewlets)', 'Foundation (1878: F10xxRC2_20090903) (RTC Development)', 'Foundation Web UI (1853: F10xxRC3_20090911b) (Web UI)', 'Install Common (45: v20090803-a) (Install)', 'Install Foundation (116: v20091014-20-a) (Install)', 'JazzDevelopmentTools (update site) (98: foundation.10xx.integration_I20090824-1700) (Matt Lavin)', 'JFS (648: F10xxRC3_20090909) (Repository)', 'Process (2907: F1001RC3_20090908) (Process)', 'Process UI (2578: F1001RC2_20090902) (Process)', 'RelEng (6222: NL_changes_20090908-1215) (RTC Development)', 'RelEng Common (1: Initial Baseline) (Build Framework)', and 'RelEng Foundation (1: Initial Baseline) (Release Engineering)'.



The developer's Jazz workspace

- Each developer has her own Jazz workspace
- All work is done initially in the developer's Jazz workspace
- The developer's Jazz workspace "flows with" the integration stream
 - ▶ When the Jazz workspace is created, its contents match the contents of the integration stream at the time of the creation of the workspace
 - ▶ "Accept" operations transfer changes from the integration stream to the developer's Jazz workspace
 - ▶ "Deliver" operations transfer changes from the Jazz workspace to the integration stream

Note that a Jazz workspace is not the same as an Eclipse workspace.



A sample Jazz developer's workspace

The screenshot displays the 'Repository Workspace' configuration page in IBM Jazz. The workspace name is 'Pete's Build Framework 3.0 - Foundation Stable Workspace'. The repository is 'psteinfe@jazzdev.torolab.ibm.com', owned by Peter Steinfeld, with public visibility. A list of components is shown, including Dashboard, Foundation, and various Eclipse prerequisites. The 'Flow Targets' section lists 'Build Framework 3.0 - Foundation Stable (default) (current)', 'David Olsen Foundation', and 'Foundation Integration 2.0'. Callouts highlight the workspace name, the component list, and the flow target.

Repository Workspace Save

Name:* Pete's Build Framework 3.0 - Foundation Stable Workspace

Details

Repository:* psteinfe@jazzdev.torolab.ibm.com Browse...

Owned by:* Peter Steinfeld Change...

Visibility: Public

Description:

Components

Shows the components in this repository workspace.

- Dashboard (917: 1001_RC3_20090911b) (Dashboard Viewlets)
- Foundation (1878: F10xxRC2_20090903) (RTC Development)
- Foundation Web UI (1853: F10xxRC3_20090911b) (Web UI)
- Install Common (45: v20090803-a) (Install)
- Install Foundation (116: v20091014-20-a) (Install)
- JazzDevelopmentTools (update site) (98: foundation.10xx.integration_I20090824-1700) (Matt Lavin)
- JFS (648: F10xxRC3_20090909) (Repository)
- Prereqs - Common (1: Initial Baseline) (Sonia Dimitrov)
- Prereqs - Eclipse CVS Client (1: Initial Baseline) (Sonia Dimitrov)
- Prereqs - Eclipse JDT (1: Initial Baseline) (Sonia Dimitrov)
- Prereqs - Eclipse PDE (1: Initial Baseline) (Sonia Dimitrov)
- Prereqs - Eclipse Platform (1: Initial Baseline) (Sonia Dimitrov)
- Prereqs - Eclipse SDK (1: Initial Baseline) (Sonia Dimitrov)
- Prereqs - Eclipse Test Framework (1: Initial Baseline) (Sonia Dimitrov)

Load component additions on save.

Flow Targets

Shows the workspaces and streams which flow from this repository workspace.

- Build Framework 3.0 - Foundation Stable (default) (current)
- David Olsen Foundation
- Foundation Integration 2.0

Buttons on the right: New..., Add..., Remove, Rename..., Change Owner..., Replace With..., Show Repository Files, Add..., Remove, Edit...



Jazz streams, workspaces, and files

- Different versions of the same file can exist –
 - ▶ On the developer's workstation
 - ▶ In the developer's Jazz workspace
 - ▶ In the integration stream
- At the moment a file is modified and saved, a new, different version exists on the workstation (unresolved).
- When the file is checked in, that version is transferred to the Jazz workspace (outgoing).
- When the change is delivered to the integration stream, that version will exist in the integration stream (incoming to other workspaces).
- The "Pending Changes" view shows the file states.



Using the Pending Changes view

1 unresolved local, 1 incoming change set, 2 outgoing change sets, 18 component changes

- Process
- Process UI
- RelEng
- RelEng Common
 - Incoming
- RelEng Foundation
 - Unresolved
 - Outgoing
 - Suspended
- RelEngBuilder
- RelEngNLS
- Repository
- Repository Examples
- Repository Gateway
- Repository Migration Tooling

Changes in the integration stream, but not in the Jazz workspace

Changes on the workstation not yet checked in

Checked in changes in the workspace that have not yet be delivered to the integration stream



Setting up a Jazz build

- The build has its own Jazz workspace, just like a developer's Jazz workspace.
- You must create a script to do the actual build.
 - ▶ The build should produce the artifacts that will be delivered to the customer
 - ▶ The build should run the automated tests
- The build runs on an build machine
 - ▶ Multiple machines can be used for efficiency
- Schedule the build –
 - ▶ Schedule a continuous build for small teams
 - Continuous builds only run if new changes have been delivered
 - ▶ Schedule fixed build times for large teams and long builds
 - Fixed times enable developers to plan coordinated deliveries



A sample Jazz build definition -- overview

□ Build Definition ▾
Save

ID: foundation.ccb

Project or Team Area: Build Framework

Browse...

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.

- apollobuildserv
- apollobuildserv10
- apollobuildserv11
- apollobuildserv12
- apollobuildserv2
- apollobuildserv3.ottawa.ibm.com
- apollobuildserv4
- apollobuildserv5
- apollobuildserv6
- apollobuildserv7

Select All

Deselect All

Create 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.

Prune build results

Successful build results to keep:

Failed build results to keep:

Overview
Schedule
Properties
Jazz Source Control
Ant

Build name

List of build engines that might run this build

This is the build Overview



A sample Jazz build definition -- scheduling

Build Definition Save

ID: Project or Team Area: Browse...

Schedule
Schedule for automatically running this build.

Enabled

Build Time

Continuous interval in minutes: This is a continuous build that starts every 257 minutes

At: :

Build Days

<input checked="" type="checkbox"/> Monday
<input checked="" type="checkbox"/> Tuesday
<input checked="" type="checkbox"/> Wednesday
<input checked="" type="checkbox"/> Thursday
<input checked="" type="checkbox"/> Friday
<input checked="" type="checkbox"/> Saturday
<input checked="" type="checkbox"/> Sunday

Runs every day

This is the build Schedule

Overview | Schedule | Properties | Jazz Source Control | Ant



A sample Jazz build definition – source control

Build Definition Save

ID: Project or Team Area: 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:* Select... Create...

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

Load Options

The versions of files in the build always match the workspace

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".

Don't build if there are no new deliveries

Overview | Schedule | Properties | Jazz Source Control | Ant



A sample Jazz build definition – build script

□ Build Definition ▾ Save

ID: Project or Team Area: Browse...

Build File and Targets
Specify the Ant build file and the targets to be invoked. Properties can be referenced using `${propertyName}`.

Build file:*
This can be an absolute path on the build machine, or may be relative to the current directory of the build engine process.

Build targets:
The targets in the build file to execute. Multiple targets must be separated by a comma. If none are specified, the build file's default target is executed.

▶ **Ant Configuration**

Overview | Schedule | Properties | Jazz Source Control | Ant

The script for the build



Running the builds

- Builds are run automatically or can be submitted on demand.
 - ▶ Continuous automatic builds only get run if new deliveries have been made since the last build.
- Each build run produces a build result
 - ▶ The build result has a clear pass/fail status
 - ▶ Build result contains log files
 - ▶ Build results contains links to artifacts produced by the build
 - ▶ Build results report test results

Build results are universally visible



Sample Jazz build results view

The screenshot shows the 'Builds' view in the Rational software interface. The 'Builds' tab is highlighted in the top navigation bar. Below it, a table lists various build entries with their status, labels, progress, and start times. Callouts provide context: 'Build results view' points to the 'Builds' tab; 'The build result for the highlighted build is displayed in its own view' points to the selected row; 'A check mark means the build succeeded' points to a green checkmark icon; and 'A red "X" means the build failed' points to a red 'X' icon.

Build	Label	Progress	Estimated Co...	Start Time
foundation.ccb (personal build by David Olsen)	RJF-T20091030-2030	Completed		October 30, 2009 8:30:46 PM
foundation.ccb	RJF-I20091030-1609	Completed		October 30, 2009 4:10:01 PM
foundation.ccb (personal build by David Olsen)	RJF-T20091030-1552	Completed		October 30, 2009 3:52:40 PM
foundation.ccb (personal build by David Olsen)	RJF-T20091030-1409	Completed		October 30, 2009 2:09:07 PM
foundation.ccb (personal build by Peter Steinfeld)	RJF-T20091030-1257	Completed		October 30, 2009 12:57:08 P
foundation.ccb	RJF-I20091030-1152	Completed		October 30, 2009 11:52:39 ..
foundation.ccb (personal build by Peter Steinfeld)	RJF-T20091030-1132	Abandoned		October 30, 2009 11:32:43 ..
foundation.ccb (personal build by Simon Archer)	RJF-T20091030-1002	Completed		October 30, 2009 10:02:55 ..
foundation.ccb (personal build by Peter Steinfeld)	20091030-1002	Completed		October 30, 2009 10:02:41 ..
foundation.ccb (personal build by David Olsen)	RJF-T20091030-0911	Completed		October 30, 2009 9:11:11 AM
foundation.ccb (personal build by Simon Archer)	RJF-T20091030-0803	Completed		October 30, 2009 8:03:16 AM
foundation.ccb (personal build by David Olsen)	RJF-T20091030-0736	Completed		October 30, 2009 7:36:20 AM



A sample Jazz build result

Build foundation.ccb RJF-I20091030-1609

Completed
 Duration: 1 hour, 15 minutes, 59 seconds
 Start Time: October 30, 2009 4:10:01 PM
 Completed: October 30, 2009 5:26:00 PM

Status Trend:

Reported Work Items
 None reported against this build
[Create a new work item](#)
[Associate an existing work item](#)

Contribution Summary

- Downloads: [9 downloads](#)
- External Links: [2 links](#)
- Logs: [10 logs](#)
- Snapshot: [foundation.ccb_20091030-1609](#)
- JUnit: [5317 tests, 22 failures, 3405 errors](#)
- Repository Workspace: [BuildFoundationExperimental](#)
- Work items: [1 included in build](#)
- Changes: [Show changes](#)

General Information

- Requested by: (scheduled build)
- Build Definition: [foundation.ccb](#)
- Build Engine: [foundation.ccb.cclbld19](#)
- Build History: [46 builds](#)
- Deletion allowed

Overview | Activities | **JUnit** | Logs | External Links | Downloads | Properties



What happens during a build

- A build request is submitted, either on schedule or by user request
- For scheduled build requests with no newly delivered changes, nothing happens
- The build machines poll the Jazz server looking for build requests
- The build request gets matched to a build machine
- Changes are accepted from the integration stream to the workspace of the build
- A snapshot is created of the build's workspace
- The build machine fetches files using the versions in the snapshot
- The build script runs on the build machine
- Results are reported to the Jazz build result
 - ▶ Log files
 - ▶ Test results
 - ▶ Created artifacts



Personal builds

- Allow developers to do a test build of the contents of their Jazz workspace before delivering to the integration stream
- Similar to integration builds
 - ▶ Personal builds use the same build definition as the integration build
 - ▶ Personal builds run on the regular build machines
 - ▶ Personal builds use the regular build scripts
- But they're different ...
 - ▶ Personal builds take input from a developer's Jazz workspace rather than from the build's workspace (the integration stream)
 - ▶ Personal builds do not create snapshots



Requesting a personal build

The screenshot shows the 'Request Build' dialog box with the following fields and callouts:

- Build:** foundation.ccb (Callout: The name of the build, the same build used for the integration stream)
- Build Options:**
 - Personal Build:** (Callout: Checkbox indicates a personal build)
- Repository workspace:*** Pete's Build Framework 3.0 - Foundation Stable Workspace (Callout: The Jazz workspace from which the source code is taken)
- Component load rules:** (Empty field)
- After submitting the request:** Show builds of the requested type (Dropdown menu)

Buttons: Submit, Cancel



Daily project rhythm

- ✓ Developers deliver changes to the build
- ✓ Builds run and report results
- ✓ Broken builds get fixed immediately
- ✓ Everyone watches the builds for status
- ✓ System test engineers pick up test artifacts from the Downloads link on the build results page



Foolproof delivery process

Developer deliveries must not break the build

- ✓ Jane, our developer, accepts from the integration stream into her Jazz workspace – now her Jazz workspace matches the integration stream
- ✓ Jane develops new code and automated tests in her own Jazz workspace
- ✓ Meanwhile, other developers deliver changes to the integration stream
- ✓ When Jane is ready to deliver, she accepts changes from the integration stream into her Jazz workspace *again* and does any necessary integration
- ✓ Jane does a personal build from the files in her Jazz workspace
- ✓ If the personal build succeeds, she delivers to the integration stream
- ✓ If the personal build fails, she continues working until she gets a successful build in a workspace that contains a mix of the current integration stream and Jane's changes



Fixing broken builds

- Create a work item
- Analyze the log files
- Analyze the test results
- Look at the changes delivered since the last build
- Use the build's snapshot to reproduce the build's contents on a workstation
 - ▶ Compare the snapshot to Jane's workspace
 - ▶ Create a Jazz workspace with the same contents as the snapshot to reproduce the problem



A sample Jazz build result

Build foundation.ccb RJF-I20091030-1609

Completed

Duration: 1 hour, 15 minutes, 59 seconds
Start Time: October 30, 2009 4:10:01 PM
Completed: October 30, 2009 5:26:00 PM

Status Trend: [Progress bar]

Contribution Summary

- Downloads: [9 downloads](#)
- External Links: [2 links](#)
- Logs: [10 logs](#)
- Snapshot: [foundation.ccb_20091030-1609](#)
- JUnit: [5317 tests, 22 failures, 3405 errors](#)
- Repository Workspace: [BuildFoundationExperimental](#)
- Work items: [1 included in build](#)
- Changes: [Show changes](#)

Reported Work Items

- None reported against this build
- [Create a new work item](#)
- [Associate an existing work item](#)

General Information

- Reported by: (scheduled build)
- Build Definition: [foundation.ccb](#)
- Build Engine: [foundation.ccb.cclbld19](#)
- Build History: [46 builds](#)
- Deletion allowed

Overview | Activities | **JUnit** | Logs | External Links | Downloads | Properties



Using snapshots

The screenshot shows the 'Snapshot' tool interface. At the top, the title is 'Snapshot' and there is a 'Save' button. Below the title, the 'Name' field contains 'foundation.ccb_20091030-1609'. The interface is divided into several sections:

- Details:** Shows 'Created by: Build User Account', 'Created on: Oct 30, 2009 4:10 PM', 'Modified on: Oct 30, 2009 4:10 PM', and a 'Description' field with the text 'Snapshot created by automated build'.
- Links:** A list of actions: 'Create a new repository workspace', 'Create a new stream', 'Compare with snapshot', and 'Compare with repository workspace or stream'.
- Components:** A list of components in the snapshot: 'Dashboard (921: foundation.ccb_20091006-1611)', 'Foundation (1924: foundation.ccb_20091019-0001)', 'Foundation Web UI (1853: F10xxRC3_20090911b)', 'Install Common (65: foundation.ccb_20091014-1244)', 'Install Foundation (116: v20091014-20-a)', 'JazzDevelopmentTools (update site) (98: foundation.10xx.integration_I20090824-1700)', and 'JFS (795: foundation.ccb_20091022-1350)'. A 'Show Repository Files' button is located to the right of this list.

Four callout boxes provide additional information:

- One points to the 'Name' field: 'The snapshot name is derived from build that created this snapshot'.
- Another points to the 'Links' section: 'Create a new Jazz workspace that matches what went into the build'.
- A third points to the 'Compare with repository workspace or stream' link: 'Compare what went into the build with a developer's Jazz workspace'.
- A fourth points to the 'Components' list.



The conversation you want to have ...

Your manager

How's the project going?

What have you implemented so far?

How do you know that A, B, and C are complete?

That's great! How can I try them out?

You

We have clean builds every day.

We've completed features A, B, and C, and are working on D and E.

We have automated tests that check them in every build.

Just grab them from latest build. Let's do it right now ...

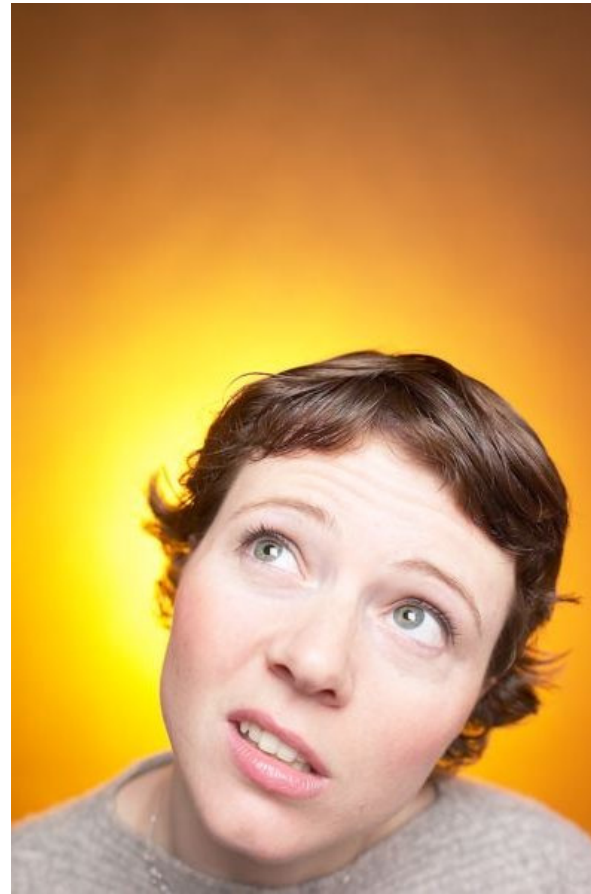


References

- jazz.net -- <https://jazz.net/index.jsp>
- Jazz build FAQ -- <https://jazz.net/wiki/bin/view/Main/BuildFAQ>
- Personal builds -- <http://jazz.net/wiki/bin/view/Main/BuildPersonal>
- psteinfe@us.ibm.com



Questions?





© Copyright IBM Corporation 2009. All rights reserved.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo, the on-demand business logo, Rational, the Rational logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.

