Jadeite
ReadMe and User's Guide

3.0.85

October 2019

doc rev. October 28, 2019

GemTalk™ SYSTEMS

This documentation is furnished for informational use only and is subject to change without notice. GemTalk Systems LLC assumes no responsibility or liability for any errors or inaccuracies that may appear in this documentation.

COPYRIGHTS

TRADEMARKS

GemTalk Systems LLC
15220 NW Greenbrier Parkway
Suite 240
Beaverton, OR 97006

Table of Contents

Chapter 1. Introduction to Jadeite

Supported Jadeite Platforms . . . 5
Supported GemStone Server Platforms . . . 5
Other Documentation . . . 5
1.1 Installing Jadeite . . . 6
Requirements for Rowan users other than DataCurator and SystemUser . . . 6
1.2 Error Reporting . . . 6
1.3 Rowan Overview . . . 6
Code management in Rowan . . . 7
Rowan Packages . . . 7
Class categories and Package names . . . 7
Rowan Projects . . . 7
Project Loading . . . 8
Class Initialization . . . 8
GemStone SymbolDictionaries . . . 8
Extension Methods . . . 9
Method category (protocol) names and Package names . . . 9
Global extension methods . . . 9
Class Versioning in Rowan . . . 9
Rowan Files, Directories, and Formats . . . 9
Rowan Project Configuration Files . . . 10
Rowan Project Specification Files . . . 10

Chapter 2. Jadeite

Jadeite vs. GBS . . . 11
Transaction mode . . . 11
2.1 Logging in and Project Operations . . . 12
Logging in . . . 12

Console Window . . . 13
The Console Jadeite menu . . . 14
Preferences File . . . 15
Jadeite settings . . . 15
Startup windows . . . 15
Preferences File Browser . . . 15
The Console Browse menu . . . 16
Project Operations and Workflow . . . 17
Project Menu Operations . . . 18
Changes Browser . . . 19
Example Project Load . . . 20
2.2 Browsing and Code Development . . . 21
The Project Browser . . . 22
Projects Pane . . . 22
Packages Pane / Dictionaries Pane . . . 23
Classes Pane . . . 23
Categories Pane / Variables Pane . . . 23
Methods Pane . . . 23
Multiple Tabs in Project Browser . . . 24
Features and Attributes of the lower/code pane . . . 24
Background Color . . . 24
Class Definition Tab . . . 24
Class Documentation Tab . . . 24
Method Source Tab . . . 24
SUnit Tab . . . 25
Comparison Tab . . . 26
Project Tab . . . 26
Method List Browser . . . 27
Querying: Senders, Implementors, etc. . . . 28
Popup Menu context-aware menu items . . . 28
SUnit Browser . . . 30
2.3 Debugger . . . 31
Stepping through code . . . 32
Breakpoints . . . 32
Breakpoint Browser . . . 33
Disabling Breakpoints . . . 34
2.4 Code Caveats . . . 35
Non-Rowanized code . . . 35
Aborts and the Transient Symbol Dictionary . . . 35
Adding Packages to Project . . . 35

Chapter 3. Recommended Workflow

Getting Started . . . 37
Code development . . . 37
Sharing work . . . 38

Chapter

1 Introduction to Jadeite

Jadeite is a graphical user interface for GemStone Smalltalk development. Jadeite is a Dolphin-based Smalltalk application that runs on Windows. It allows login to a GemStone Repository in which the Rowan tools are loaded, to allow project and package management, code development, and debugging.

Jadeite relies on the **Jadeite Rowan Services** layer, which is provided with the Rowan Tools. The coordinating version of the Jadeite Rowan Services must be loaded in the Rowan GemStone repository for Jadeite to work correctly.

If the versions of Jadeite and the Jadeite Rowan Services do not match, on login or on Rowan load, a warning is reported; while login can proceed, you are likely to encounter message not understood or other errors from Jadeite operations.

Supported Jadeite Platforms

Jadeite is supported on Windows only.

It has been tested on Windows 7 and Windows 10.

Supported GemStone Server Platforms

Jadeite has been tested with versions of Rowan installed into GemStone/S v3.2.15 and 3.2.17.

Other Documentation

Jadeite's functionality is implemented in GemStone/S 64 Bit and Rowan for GemStone.

More detailed information about Rowan is provided in the *Rowan for GemStone User's Guide*, `Rowan-UsersGuide-1.0.pdf`.

Documentation for the GemStone/S 64 Bit product can be found on the GemTalk website, gemtalksystems.com.

1.1 Installing Jadeite

Jadeite is a Windows executable (Jadeite.exe) with associated directories containing .dlls and icons.

To use, copy the directory containing these to a Windows directory and start the executable jadeite.exe. No updates to the OS path or %GEMSTONE% environment variable are required.

If you are installing over a previous version of Jadeite, be sure to delete the `.stb` and `.errors` files.

Jadeite can only log into a GemStone repository that has been Rowan-enabled.

Requirements for Rowan users other than DataCurator and SystemUser

Using Rowan to write or modify code requires permissions that would not ordinarily be given to an application UserProfile. This is required for users who will load projects or update methods, and are not necessary for users logging in to browse or execute code.

- The UserProfile must have OtherPassword privilege, as well as CodeModification Privilege.

It is no longer necessary to define the #rowanCompile variable nor to enable GsPackagePolicy.

1.2 Error Reporting

Errors or unexpected behavior in Jadeite should be reported to GemTalk for triage and analysis.

When reporting a problem, the following information should be provided:

- A **screen shot** of the error, showing the Browser you were in, and providing a description of the steps taken before the error occurred.
- If a walkback occurs, a copy of the **stack trace.**
- The **Gem log**, located in the usual location for your GemStone configuration (by default, in the home directory of the UNIX user that owns the Gem process). The Gem log includes console output from Rowan, including git commands executed.

  The **Jadeite.errors** file in the directory from which you started Jadeite, which includes errors on the Jadeite client.

1.3 Rowan Overview

Rowan provides the underlying support for code management that is used by the Jadeite tools. Rowan implements Projects and Packages to associate related classes and methods, and handle the loading, saving, and other behavior of Projects and Packages.

More details on Rowan are provided in the *Rowan for GemStone User's Guide*. This overview provides some basic introduction, to make the Jadeite functionality easier to understand.

Code management in Rowan

In Rowan, Packages and Projects are first class GemStone objects. Everything in Rowan is based on a definition, with definitions for Projects, Packages, classes, and methods.

In base GemStone, SymbolDictionaries may be used to provide code organization; these are not a primary code organization feature in Rowan. Rowan has some specific behavior requirements between Package/Projects and SymbolDictionaries.

Since Rowan provides access to Git to perform the code version control, Rowan provides support for Git branches. A Git branch is an independent development line which may have many commits before being merged with the main development branch. You may have many branches within a Git repository, but the files on disk represent the single current branch.

Jadeite provides menu items that access behavior in Rowan; all these operations can be done directly in GemStone Smalltalk code, by executing Rowan code. Git operations such as pushes and pulls can be done in Jadeite, using Rowan code, or by using Git commands on the Unix command line.

Rowan Packages

A Package contains definitions of classes, class and instance methods for those classes, and extension methods for classes that are not defined in this Package. Every Rowan-managed class and method is in a Package. Each Package has a name.

Class categories and Package names

In this version, the class category is used to map a class to a Package name. (Note that in GemStone Smalltalk, method protocols that group methods within a class are also called categories). Each class has a category, and for classes in Rowan, that category should exactly match the name of a Package. Package name to category name matching is case-sensitive.

Currently, Package names must therefore be unique within the image; a Package name should not be used in more than one different Project.

The Rowan underlying code does supports separate package name and class category. In some cases, such as when classes are "adopted" into Rowan, the category and package may be different.

Rowan Projects

A Project represents a set of Packages that are managed within a single Git repository. A single Git repository holds one Project.

Projects cannot be nested.

A Rowan project also has a a Rowan Project Configuration, which specifies further details for the Project. Project Configurations can be nested.

Project Configurations are not currently accessible in Jadeite; you must use Rowan to define and make changes to the Project Configuration.

Project Loading

Rowan load operations are atomic. One or more Projects may be loaded into the image from Git repository/s on disk in a single atomic load operation.

The load operation determines what changes to the image are needed, then compiles the changes into a private area that does not affect the behavior of the running system. Once all changes to the private area have succeeded, the changes are made live.

This version does not support control over ordering of Package loading.

Class Initialization

Class initialization is performed as the last step in the load. When the load is otherwise done, all new and changed class #initialize methods are executed. Class initialize methods that have no changes are not executed.

Class initialization execution orders superclasses before subclasses, but does not define ordering between peer classes.

You can avoid initializing classes by setting up a handling block for RwExecuteClassInitializeMethodsAfterLoadNotification. For example, the following disable class initialization for all classes loaded by the given block, or only selected classes, respectively:

```
[ Rowan projectTools load expression ]
      on: RwExecuteClassInitializeMethodsAfterLoadNotification
      do: [:ex | ex resume: false ].

[ Rowan projectTools load expression ]
   on: RwExecuteClassInitializeMethodsAfterLoadNotification
   do: [:ex |
      (listOfClasssNames includes: ex candidateClass name)
         ifTrue: [ ex resume: false ]
```

GemStone SymbolDictionaries

All classes and methods extensions defined within a single Package are loaded and related to a single symbol dictionary. The same SymbolDictionary may be used by multiple Packages and multiple Projects.

A Project and Project Configuration have a default SymbolDictionary, which will be used for all classes and methods in all Packages unless otherwise specified.

The Project Configuration allows each individual Package's classes and methods to be loaded into a specific SymbolDictionary. This can be decided conditionally at load time, based on the user or other conditional attributes.

Managing the organization of code in SymbolDictionaries is not yet handled by Jadeite, but can be done by executing Rowan code and by editing the Configuration files on disk.

The Jadeite Project Browser provides a way to view the SymbolDictionaries in your image, and the classes contained in them, by selecting the project (NONE).

Extension Methods

In base GemStone, each class and instance method follows its class. By default, the methods in a class are defined in the same Package as the class itself.

However, methods may belong to a class that is defined in a different Package. This is done, in this version, by putting the method in a method category (protocol) with a specifically formatted name, as follows.

Method category (protocol) names and Package names

To define a method in a package other than the package that its class is in, you will add the method to a specifically-named method category (method protocol), composed of a leading asterisk followed by the name of the package.

'**PackageName*'

For example, if there is a Package named Foo-Bar, all methods in a method category named *Foo-Bar will be in the Foo-Bar Package, no matter which Package the method's class is defined in.

This limits the flexibility in organizing methods by method category, since moving a method into another category may change its package location.

Package name matching for extension method categories is case-insensitive (unlike the matching for class category name).

Global extension methods

Defining extension methods on base classes is a special case, since these methods must be in a package that loaded into the Globals SymbolDictionary, which is not appropriate for ordinary application classes. Currently, such extension methods should be defined by executing Rowan code.

Class Versioning in Rowan

When a new version of a class is created, all methods present in the existing version are recompiled in the new class version.

If the existing version has subclasses, a new version of each of the subclasses is created, and all methods in all these classes are recompiled.

By default, instance of versioned classes are not migrated to the new version. Automatic migration this can be setup using Rowan protocol.

When one or more a new class versions are created by a Rowan load, these class versions and the versions of all subclasses are created within the single atomic load operation, along with any other added, removed, or modified methods and classes.

Rowan Files, Directories, and Formats

The details of Rowan are described in more detail in the *Rowan for GemStone User's Guide*.

A Rowan repository contains three important directories (in addition to other files), which are normally under the `rowan` directory:

`specs/`
: containing one or more Project specification files that specify how the project is

loaded. These files are in `.ston` format, Smalltalk Object Notation, which is an object serialization format that allows any object to be represented in a file.

`configs/`
: containing one or more Project Configuration files that specifies the packages and other Project Configurations in a Project. These files are also in `.ston` format. Project configuration files may be nested, and load different sets of the packages that are defined within the Project.

`src/`
: containing the actual source code, organized by package, in one file per class `.st` files; these are text files in tonel format. Tonel is a text based format for Smalltalk code.

Rowan Project Configuration Files

The `configs` directory includes Rowan Project Configuration Files, in ston format. These files specify package names and other Configuration files containing package names, which together define the project's contents.

Rowan Project Specification Files

The `specs` directory contains Rowan Project Loading specification files. These include all the information needed to clone and load the given project, including the Git repository information, the packages, and the user name and default symbol dictionary for loading.

While a Project Specification file normally resides in the project's Git repository, it can be copied elsewhere, and used by Jadeite tools to clone the project Git repository.

Chapter

2 Jadeite

Jadeite provides an API for working with GemStone server code that is contained in Rowan Packages and Projects.

The basic windows and menus that support the workflow are included in this User's Guide, but much of the operation is intuitive to users familiar with other Smalltalk IDEs.

The processes required to create a new Project, or move existing code into a Project, are not currently supported entirely in Jadeite. See the Rowan User's Guide for details on operations that are done programmatically. The Jadeite operations described below can all be done programmatically in Rowan using Jadeite Rowan Services.

Note that the Rowan projects are loaded as SystemUser. For full visibility of Rowan code, or to load new versions of Rowan, you will need to log as SystemUser.

Rowan for GemStone relies on Git for code management. When multiple users work on a single underlying Git code base, processes should be put in place and followed, to avoid overwriting or losing work. See Chapter 3 for recommendations.

Jadeite vs. GBS

Jadeite, unlike GBS, does not replicate server code or objects to the client. Each operation retrieves the relevant data from the server for display in the Jadeite views, and code executed in a workspace in Jadeite is executed on the GemStone server.

Transaction mode

Jadeite normally runs in automatic transaction mode, and does not provide tools support for running in other mode. You may switch to other modes by executing code such as:

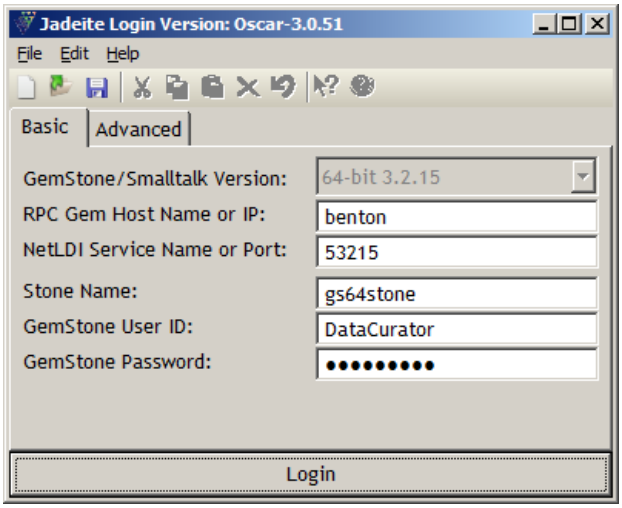
```
System transactionMode: #manualBegin
```

2.1 Logging in and Project Operations

Jadeite provides a subset of tools, similar to the tools provided by GBS or other Smalltalk IDE.

Logging in

Figure 2.1 Login Dialog



The Login Dialog is similar to the GBS Session Parameters Editor, and includes the parameters required to log in to a GemStone repository. In addition to the common parameters show, the **Advanced** tab includes additional GemStone login parameters with default values.

These parameters are stored in the Windows registry, so are initialized when Jadeite is started up again on the same node.

Only one session at a time may be logged in.

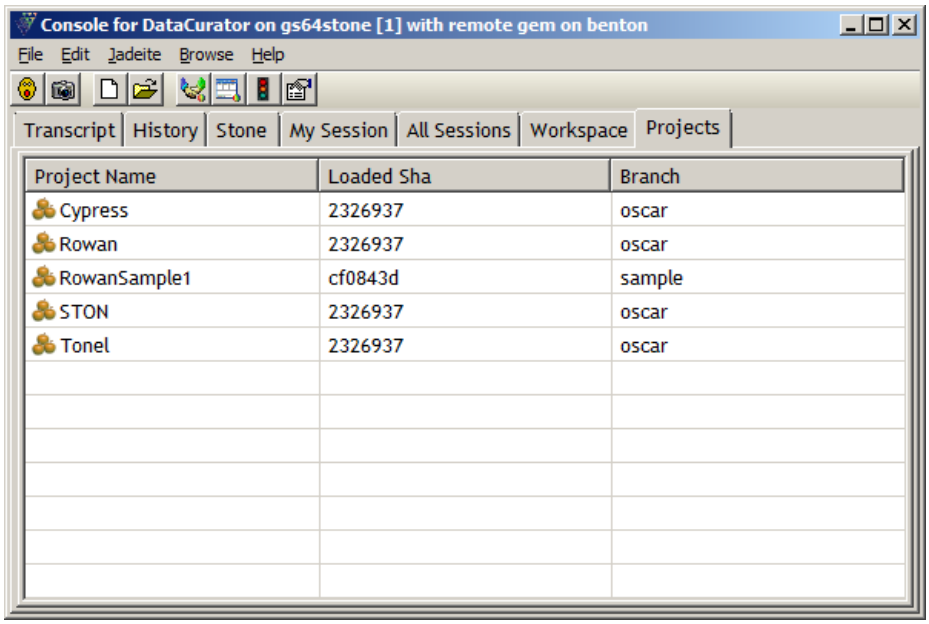
Console Window

On login, Jadeite opens a Console Window. The Console remains open for the lifetime of the session; closing the Console:

- logs out the GemStone session, asking if you wish to commit the transaction;
- closes all other open Jadeite windows; and
- returns you to the Login Dialog, re-enabling the Login button.

You may use the control-F7 button at any time to bring up the Console window.

Figure 2.2 Console



The Console provides:

- A number of tabs providing system information and workspace; and
- Menus to access system configuration settings, and to open the various browsers.

On login, the initial selected tab is the **Projects** tab.

You may use the **Workspace** tab to execute Smalltalk code; this code is executed on the GemStone server. Jadeite also supports separate workspaces, using options on the **File** menu and icon toolbar.

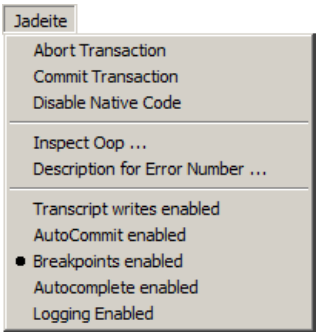
The other tabs provide GemStone system information, based on GemStone System class queries, and are primarily of use for server monitoring and administration, and debugging.

Each tab has individual pop-up menus that are specific to the functions of that tab.

The menu bar menus provide system-wide functionality, regardless of the selected tab.

The Console Jadeite menu

The Jadeite menu on the Console window includes basic GemStone operations such as commit, and allows you to enable and disable some system-wide features.



The preference features that can be disabled or enabled using this menu are listed below. A black circle means the feature is enabled; without a black circle, the feature is disabled.

These options are not persistent across logins; however, you can configure these in the preferences files (described on page 15), so they are set during login.

Transcript writes enabled

When enabled, the Jadeite transcript will be updated with Rowan commands. Rowan performs transcript writes using client forwarders.

Over a slower network, these may impact performance for many operations, such as opening browsers.

AutoCommit enabled

When enabled, a GemStone commit (not a git commit) is performed after every successful server operation. This includes both method saves and ad-hoc executions that create or modify persistent objects.

When autocommit is enabled, the camera icon in the lower left corner of the Project Browser and Method List Browser is green (see graphic on page 22)

Breakpoints enabled

Using this, you can enable and disable all breakpoints without specifically removing them.

Autocomplete enabled

When enabled, Jadeite will attempt to determine the word you are typing and make suggestions.

Logging Enabled

When logging is enabled, underlying dolphin walkback stack traces are written to a log file, which may be used for problem diagnosis by GemTalk Engineering. The logs can be browsed using the **Browse** menu item **Browse Logs**.

Enabling logging impacts performances; it is recommended to leave this disabled except when specifically diagnosing a problem.

Preferences File

Jadeite has the option to read preferences from a disk file on startup. A file named `Jadeite.prefs`, located in the same directory as the Jadeite executable, is read on login, and used to configure preferences and trigger the opening of specific browsers. If the file does not exist, default values are used.

You may use the Preferences Browser to create a file based on the template, or reset the settings.

The preferences file `Jadeite.prefs` is a simple text file that may contain one or multiple of the directives listed in this section. A leading # can be used to include text comments, or to comment out and disable specific settings.

The options that can be set fall into two groups.

Jadeite settings

These are system-side Jadeite options that can also be set from the Console Jadeite menu:

`autocommitEnabled:` *true* | *false* (Default is **false**)
`autocompleteEnabled:` *true* | *false* (Default is **true**)
`breakpointsEnabled:` *true* | *false* (Default is **true**)
`loggingEnabled:` *true* | *false* (Default is **false**)
`transcriptEnabled:` *true* | *false* (Default is **false**)

If a setting is present more than once in the file, the value of the last one is used.

If the line is commented out, or not present, the system default value is used.

Startup windows

These specify Browser Windows that can be automatically opened on login. For example, you may wish to always open a Project Browser on a particular Project.

The following types of windows can be set to automatically open:

`openBrowserOnClass:` *className*
`openBrowserOnPackage:` *packageName*
`openBrowserOnProject:` *projectName*
`openSUnitBrowserOnPackage:` *packageName*
`openWorkspace:` *pathAndfileName*

These entries may appear more than once, so you can open more than one Browser window of the same type or multiple workspaces.

If the argument does not exist, the specified Browser window will be opened, but no selection made.

Preferences File Browser

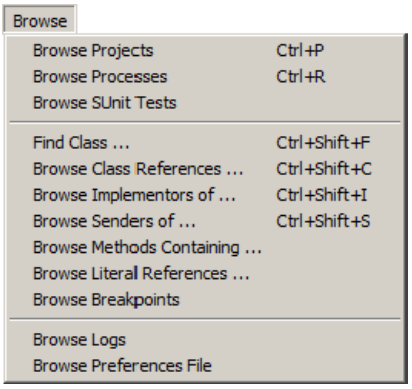
The Console Browser menu item **Browse Preferences File** brings up the Preferences File Browser, similar to a workspace. This adds the following to the **Jadeite** menu:

Preferences Options Workspace
This opens a workspace containing example default preferences.

Reset Preferences to default
This resets the text in the workspace to the default. You must save this for the changes to take effect on the next login.

The Console Browse menu

The **Browse** menu on the Console window includes options that open the various Jadeite browsers.



The menu items include:

Browse Projects
Open the Projects Browser.

Browse Processes
Open a list of processes within the current session.

Browse SUnit Tests
Open the SUnit Browser, allowing you to examine and run all SUnit Tests in the system.

Find Class...
Prompt for the name of a class, providing a drop-down list of matches, and open a Projects Browser with that class selected.

Find Class References...
Prompts for the name of a class, providing a drop-down list of matches, and opens a Method List Browser containing all methods that have a reference to that class.

Browse Implementors of ...
Prompts for a method selector, and open a Method List Browser on all implementors of that selector.

Browse Senders of ...
Prompts for a method selector, and open a Method List Browser on all senders of that selector. You may include a trailing * to perform a wild-card search.

Browse Methods Containing ...
Prompts for a string, and open a Method List Browser on all methods that contain that sequence of characters. You may include a trailing * to perform a wild-card search.

Browse Literal References ...
Prompts for a literal reference, and open a Method List Browser on all methods that reference that literal. Literal references include numbers such as SmallIntegers and SmallDoubles, literal Arrays such as #(1); and strings (enclosed in single quotes) and symbols (with leading #).

Browse Breakpoints
Opens the Breakpoint Browser, a Method List containing all methods that have breakpoints.

Browse Logs
Opens the Jadeite Log Browser. If logging was enabled, this log contains logging information for debugging.

Browse Preferences File
Opens the preferences file for Jadeite, or creates a template if it does not already exist.

Project Operations and Workflow

Projects are the highest level of code organization in these versions of Jadeite and Rowan. All application code is organized in Projects.

When you login using Jadeite, the Console displays the Projects tab. This gives you an immediate overview of the state of your projects. The same list of Projects is also shown in leftmost pane in the Project Browser.

Projects are displayed with text changes indicating the following state:

- Projects that are loaded but have no changes either in the image or on disk are in normal font.
- If there are changes to the on-disk code for a Project that are not loaded in the image (skew), the project is in Red font.

  Red does not necessarily indicate a need for action. However, **do not write to disk** if there is skew, since it may create corruption in the logical state in the git repository itself.

  You may select multiple projects and load them in one operation on the Console Projects tab.

  If you have skew on the Rowan Projects, you must login as SystemUser in order to update.
- A Project that is attached (the **Clone Git Project...** menu item was used), but not loaded, is displayed in ***Bold Italic font***. This will not have a "Loaded Sha" number (the middle column on the Console Projects tab). The loaded Sha is a column in the Console Projects tab; to see this in the Project Browser you can use the lower pane Project tab.
- A Project that has changes in the image that have not been written to Git is in ***Bold Italic font***.
- The default Project is displayed with an asterisk *.

The Console Project Tabs is multiple-select; this allows you to load or write multiple Projects in one operations. The Project Browser Project Pane is single select. The Project Browser Project menu, and the popup menus for the Project Pane and the Console Project tab, have similar options.

Workflow operations can be done either in the Console Projects tab, or in the Projects Browser. For more on the recommended workflow, see Chapter 3.

Project Menu Operations

The following workflow project operation menu items are available:

Make Default
Make the selected Project to be the default Project.

Clone Git Project...
Opens a dialog requesting the url for a Rowan specification .ston file, and a disk path for the Git repository to be cloned (copied).

If the Project has not already been cloned to this location, the Project specified by the Rowan .ston file is cloned to the given directory. If it is present, the existing clone is not affected, and the result is only that the Project is "attached" to the GemStone repository. This operation does not load the project.

Load
Load the selected Project or Projects from disk. This is used to load an attached Project, or to update an existing loaded Project. Any existing code for this Project in the image is reverted to the on-disk version of the Project.

Unload
Unload the selected project from the Repository. The project will remain on disk.

Pull from Git
Updates your local Git repository for the selected Project from the shared Git repository.

Commit to Git...
Writes out the code in your image for the selected project to disk, and commits it to the local Git repository. A commit comment is required.

Note that this does NOT perform a GemStone commit. It is recommended to perform GemStone commits frequently to ensure your work is saved.

Push to Git
Pushes the project in your local Git repository to the shared Git repository.

Changes
Display the changes in the selected Project (further detail on page 19).

Write
Writes out the code in your image for the selected project to disk, but does not commit to Git. This allows you to perform command line Git diffing before committing to Git.

Checkout Git Branch...
Opens a dialog listing the current branches in your repository, and give you to the option to checkout (switch) to a branch.

Git Log
Display the Git commit log for the selected Project.

Browse Commit In Github
For Projects cloned from GitHub, opens a Browser window on GitHub with the given commit.

Refresh
Update the display.

Note that these menus have a few additional menu items that are not common to both.

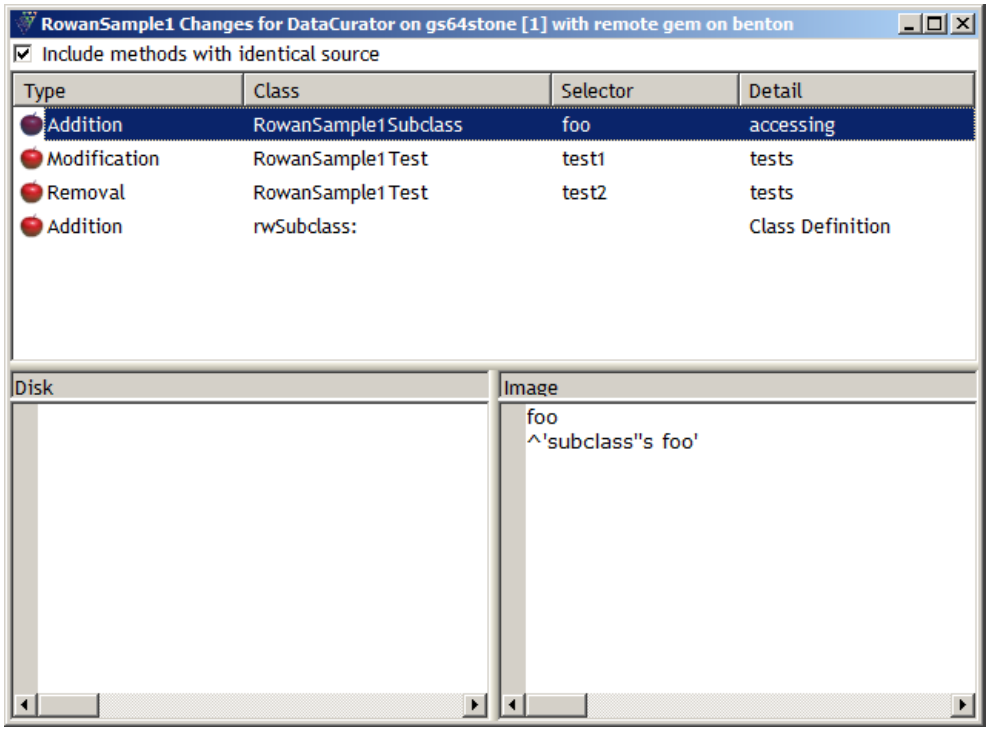
Changes Browser

Before writing your work to disk, or committing it to git, you may wish to review the set of changes you've made. The Changes Browser provides a list of all changes between what is on disk, and the image.

The Changes Browser can be opened from the Project menu item **Changes**.

You may also see changes in just a specific package within your project, using the Project Browser Packages pane **Show Changes** menu item.

Figure 2.3 Changes Browser



The lower panes of the Changes Browser provide a comparison between the on-disk version from the git repository, and the version currently in the image.

The Changes Browser is read-only, but the Browse menu item allows you to open a browser on a selected change.

Example Project Load

The Rowan Project includes several examples, RowanSample*N*, which can be used to experiment with loading a Git repository into your image and making changes.

These are projects on github, which are provided along with the distribution as git repository directories; the specification files are in the Rowan `samples` directory.

1. Clone the Git Project

If you do not have access to GitHub, copy or clone the Git repository directories for RowanSample1 into your $ROWAN_PROJECTS_HOME directory. As long as the Git repository is present here, the load will proceed without using GitHub.

```
cd $ROWAN_PROJECTS_HOME
git clone sharedRowanSample1GitRepositoryDirectory
```

You can skip this step to load the project directly from GitHub; for the RowanSample*N* projects, this can be done since the specification files are included in the Rowan project examples directory.

1. Attach the Git Project

Use the Projects Browser menu item **Clone Git Package...**. This will clone the project from GitHub only if it is not present; if the Git repository is present, it is attached to the image.

The menu item presents a dialog with two fields: a Rowan specification file, and a directory into which to clone the repository. The default settings will clone, or verify the existence of, RowanSample1, at the location $ROWAN_PROJECTS_HOME.

```
file:$ROWAN_PROJECTS_HOME/Rowan/samples/RowanSample1.ston
```

```
$ROWAN_PROJECTS_HOME
```

2. Load the Project into your image

The Project will appear in the Projects Browser, in italic font since it is attached but not loaded.

Select this Project and use the menu item **Load...** to load it.

The font in the Projects Browser becomes normal, and the SHA of the commit is displayed, since a specific commit is loaded.

3. Select or create an individual branch

It is optional, but may be helpful to follow the workflow if you have your own branch. The Git command line will create a branch in your local repository:

```
cd $ROWAN_PROJECTS_HOME/RowanSample1
git branch myName
```

4. Checkout your branch

In the Projects Browser, use the **Checkout...** menu item to switch to your branch.

5. Commit the GemStone Transaction

Use with Transcript Window **Commit Transaction** menu item to commit your state.

2.2 Browsing and Code Development

There are two browsers that are the primary places to browse and write code:

- the Project Browser, which allows you to view both code based on the Project and Package, and code that is not packaged.
- the Method List Browser, which displays a list of methods returned from a query, such as **Implementors of** or **Senders of**.

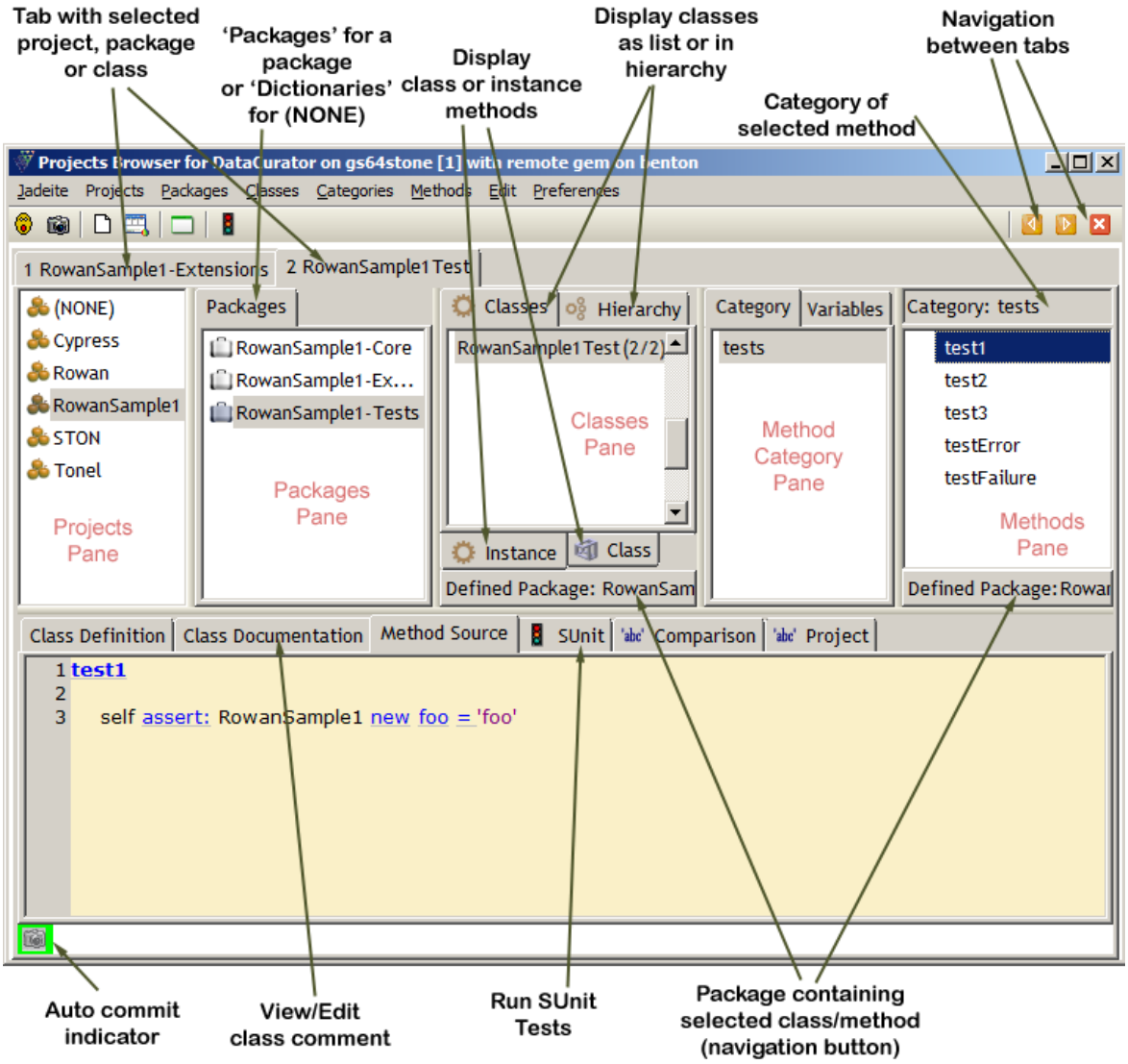
In this release, the Dictionaries Browser is no longer needed or available. The Project Browser allows you to view and edit both packaged and unpackaged code based on the SymbolDictionary.

You may also edit and save method changes in the Debugger, which provides many of the features of the Project and Method List Browser text panes.

The Project Browser

The primary browser for code development is the Project Browser.

Figure 2.4 Project Browser



The panes in this browser will be generally familiar to Smalltalk developers.

Projects Pane

Projects are listed in the Projects pane, and the operations that can be done on Projects is described in "Project Operations and Workflow" on page 17.

Rowan Projects, and Application-specific Projects, are listed by name. The package (NONE) provides allows you to browse code that is not in projects, such as base GemStone

code. The (NONE) package provides browsing by SymbolDictionary, which allows you to browse both base GemStone code and Rowan and Application code that by their SymbolDictionary.

In addition to the Project menu operations listed on page 18, the Projects menu includes the **Add Package...** option. Note that you must manually update the specification, as described under "Adding Packages to Project" on page 35; otherwise the added package is not permanently part of the project, and will be lost if the project is reloaded.

Packages Pane / Dictionaries Pane

When a project is selected, this pane lists the Packages in the selected Project or Projects. You may select multiple Packages, which will display all the classes in the selected packages in the classes pane.

When the (NONE) project is selected, this pane lists all SymbolDictionaries defined in this GemStone image. GemStone kernel code is under the Globals SymbolDictionary, Rowan classes are in Globals and in the Published SymbolDictionary,

Classes Pane

You may view the classes in the selected package alphabetically, or within the class hierarchy, depending on if the **Classes** or **Hierarchy** tab is selected

If the package contains extension methods on classes that are not in the selected package/s, those classes are shown in purple font.

If a class is selected in the **Classes** tab before you select the **Hierarchy** tab, it will show only the hierarchy for the selected class. If no class is selected, it shows when you select the **Hierarchy** tab it will show the hierarchy for all classes in the selected package.

Categories Pane / Variables Pane

Categories with names beginning with an asterisk * contain methods that reside in the package of that name (case-insensitive).

There are some exceptions, for non-Rowanized GemStone base classes. These may contain categories beginning with asterisk, although they are not associated with a project or package.

Note that categories are not first-class objects in Rowan. Category changes do not show up in the Changes Browser and may not be removed by reload. Categories are written as part of the method definition in Rowan.

With the **Variables** tab selected, you may add accessor methods for instance variables, and locate methods that read or write a particular instance variable.

Methods Pane

The methods pane lists the methods in the selected class, and in the selected category or that references the selected variables; or all methods if no category or variable is selected.

Class or instance method display is based on the **Instance / Class** tabs at the bottom of the classes pane.

All methods for the selected class are included, regardless of which package they are in or if they are not packaged.

Methods that are in a different package than the package that contains their class (extension methods), are in purple font, regardless of the selected package or packages.

Purple extension methods that not in the currently selected package/s are also underlined.

Note that, if the class is not defined in the selected package/s, and is therefor in purple font in the classes pane, the methods that are in normal font are therefore not in the currently selected package. These methods are in the same package as their class, and are not extension methods.

Normal font is also used for non-rowanized GemStone base class methods.

Multiple Tabs in Project Browser

The Project Browser supports multiple tabs, using the Project Browser's **Jadeite** menu item **New Projects Browser** or the corresponding toolbar icon. This allows you to easily move between methods, classes or projects within the same window. The tabs will automatically be labeled with the selected Project, Package or Class name.

Orange buttons on the right side, with arrows and "x", allow you to navigate between tabs or close a tab.

Features and Attributes of the lower/code pane

The lower pane provides a number of functions, such as displaying Project details and source code. There are several useful features in this pane.

Background Color

By default, the background of lower source code is light yellow. For tabs with editable text, the background becomes light green when there are unsaved changes. If there are compile errors when saving method source code, the background becomes pink.

You may use the **Preferences** menu items **Color for No Edits...**, **Color for Unsaved Edits...**, and **Color for Compile Errors...**, to define different colors.

Class Definition Tab

This allows you to view or edit a class defintion, or define a new class.

Class Documentation Tab

This allows you to view or edit the class comment.

Method Source Tab

This allows you to view and edit the source code for a method, or create a new method.

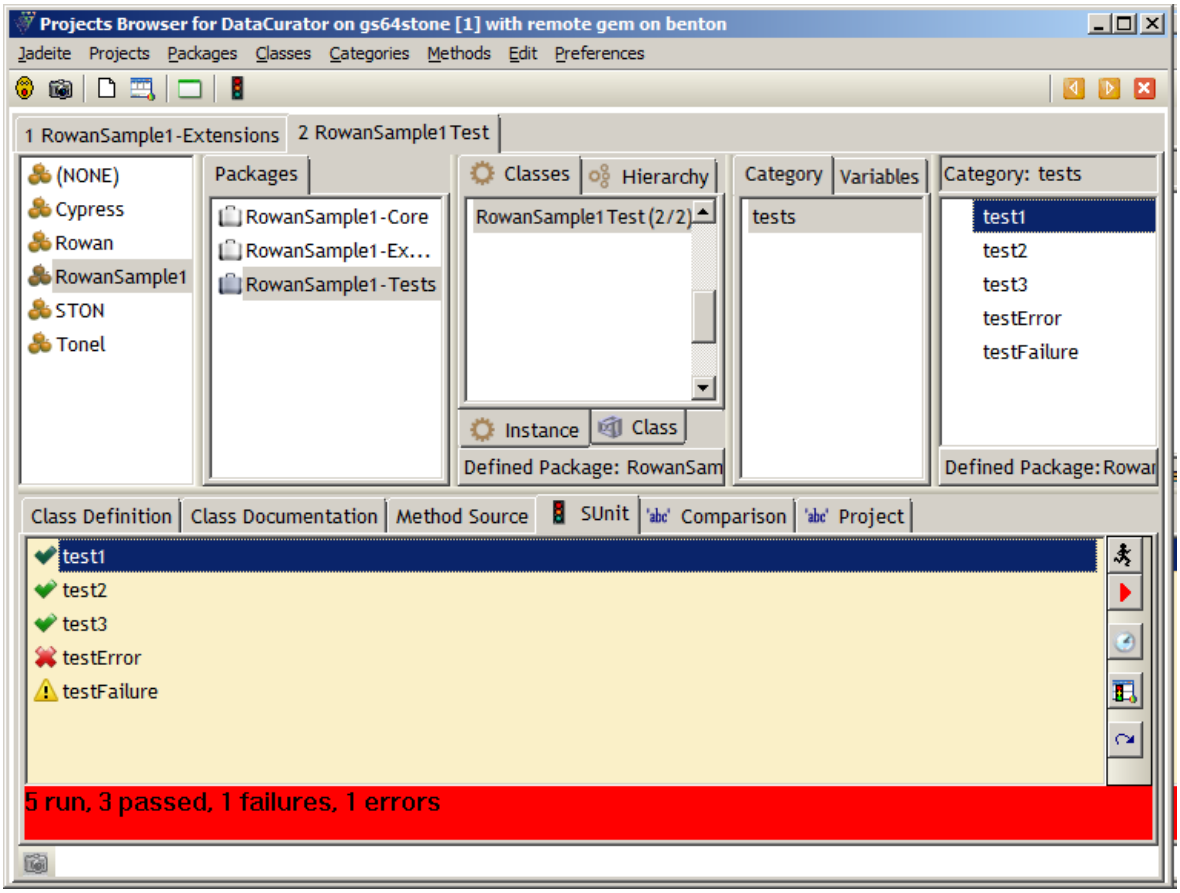
SUnit Tab

GemStone supports the SUnit testing framework. SUnit application test classes are classes defined in the hierarchy of base GemStone's TestCase class. The SUnit tab allows you to see and run all or selected tests on an test class.

Similar testing behavior is also provided in the SUnit Browser (described on page 30).

You may also run all or individual tests using method pane menu items.

Figure 2.5 SUnit tab

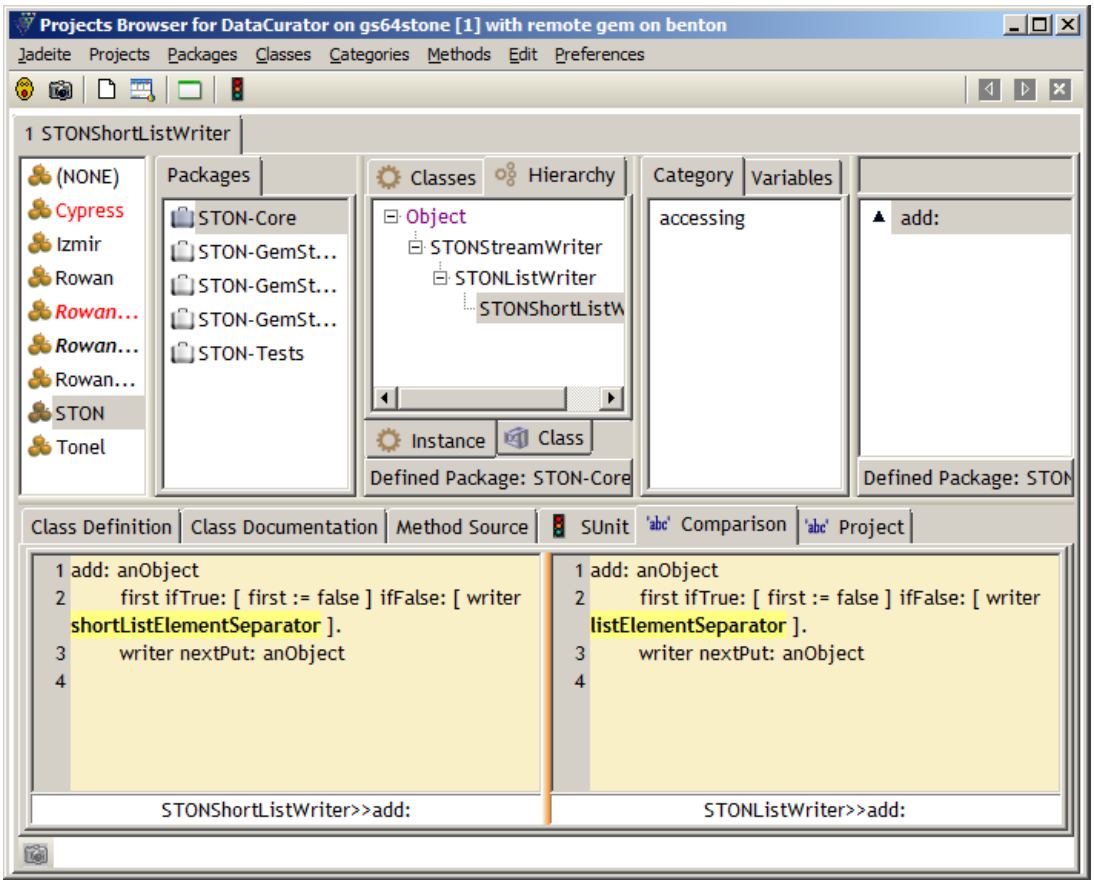


Comparison Tab

With a method selected, you may use the Comparison tab to see the differences between the selected method's implementation, and the implementation of this selector in the class's superclass hierarchy.

Text differences are highlighted in yellow.

Figure 2.6 Comparison Tab



Project Tab

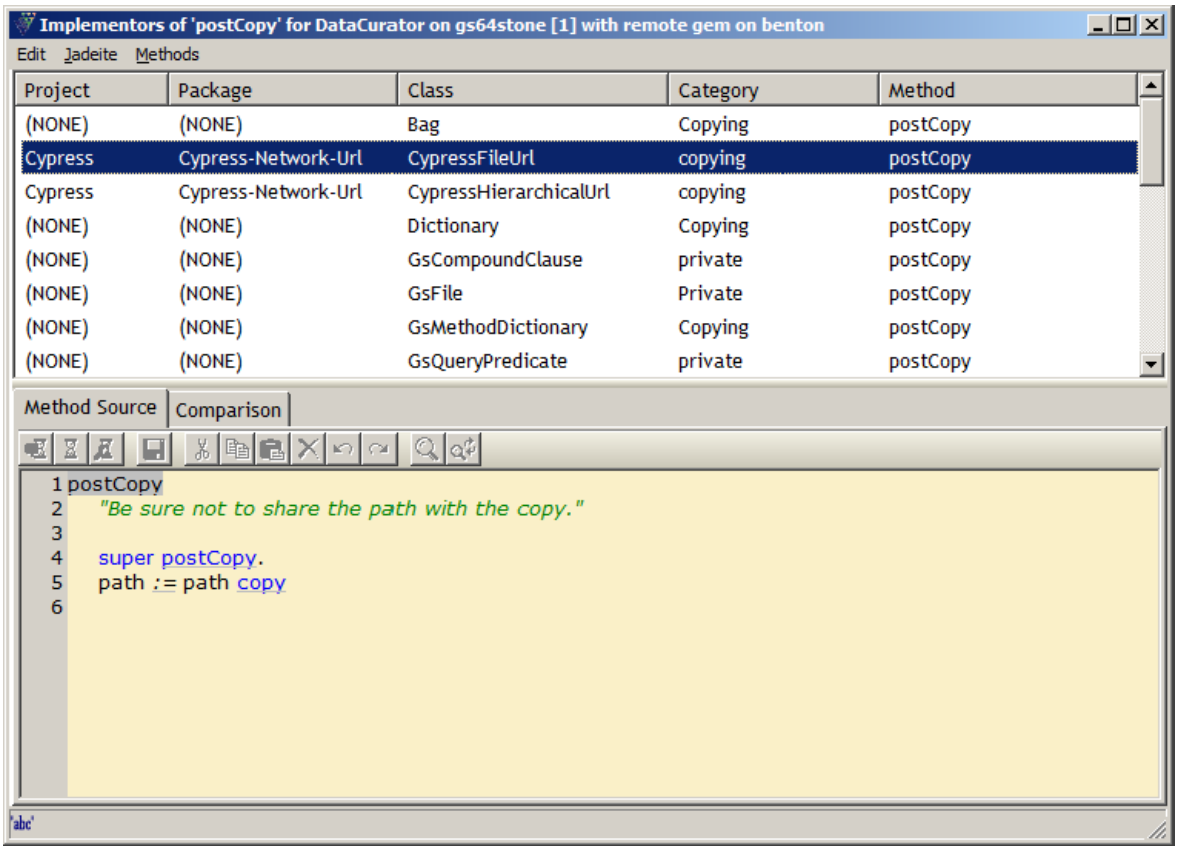
This tab provides details on the selected project.

In addition to the branch and sha, this allows you to see the projectUrl and the setting for $ROWAN_PROJECTS_HOME.

Method List Browser

When you perform operations such as **Senders Of...**, **Implementors Of...**, a method list browser is opened.

Figure 2.7 Method List Browser



The Method List Browser displays both packaged and non-rowanized base methods. Each of the column headers (Project, Package, Class, Protocol, and Method), are sortable.

Querying: Senders, Implementors, etc.

Jadeite provides a number of ways to lookup methods and classes.

Method lookup methods are found on the Project Browser methods pane. These include both lookup on the specific selected method, and lookups that query for the criteria and search over all methods in the system.

The Project Browser **Jadeite** menu provides a **Find Class...** menu item.

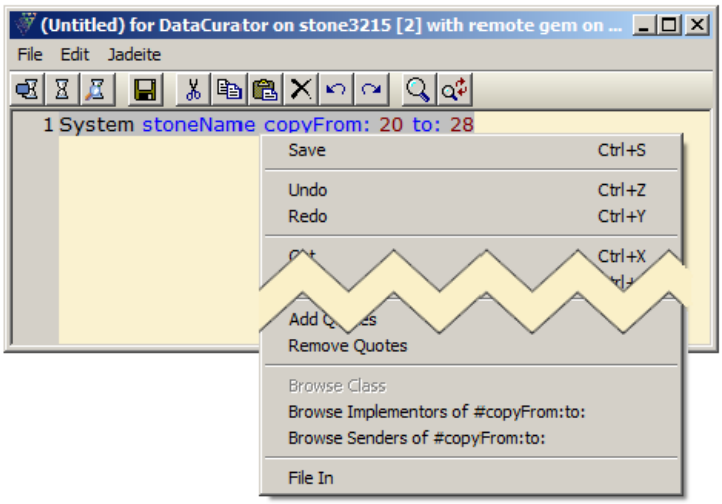
The Console **Browse** menu also provides general method lookup and find class.

Popup Menu context-aware menu items

In addition to the static method pane and drop down menus, the method source pane and workspaces will provide context-aware menu options. The syntax is somewhat different between the two; the method source knows exactly which specific compiled method selector is in use, while the workspace must rely on textual clues.

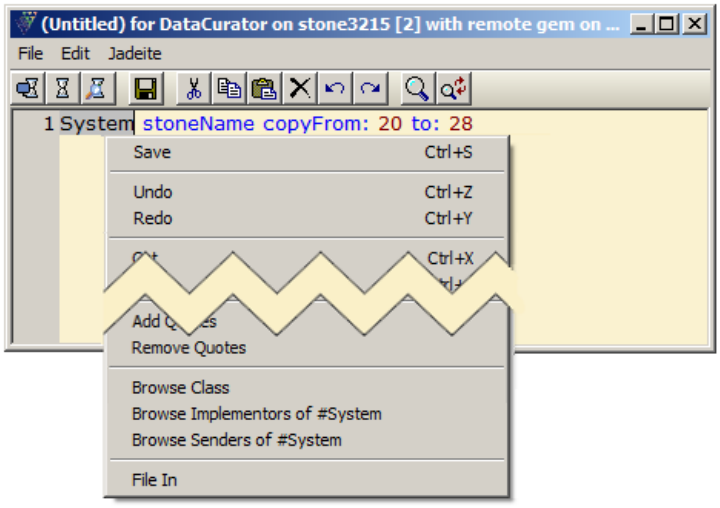
In a Workspace, you must select an expression in order to see the methods send. Note that this detects the last method send, which is a multi-keyword method.

Figure 2.8 Workspace Menu with expression selected



You can lookup a class by selecting text containing the name of a class or global in the workspace; note that **Browse Class** is now enabled:

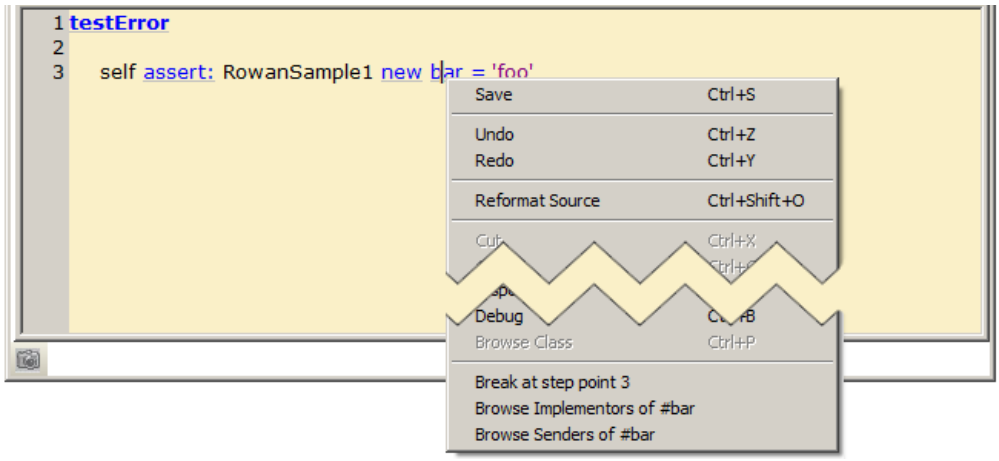
Figure 2.9 Workspace Menu with Class selected



When Browsing method source code using the Project Browser or Method List Browser, however, you do not need to select the method. Wherever the mouse point is when the menu is opened will determine the selector displayed in the senders/implementors menu items.

Note that for multiple-keyword methods, only the first keyword of the selector will be detected; if the cursor is on the second and later keyword, nothing will appear in the menu.

Figure 2.10 Method Source Menu without selection

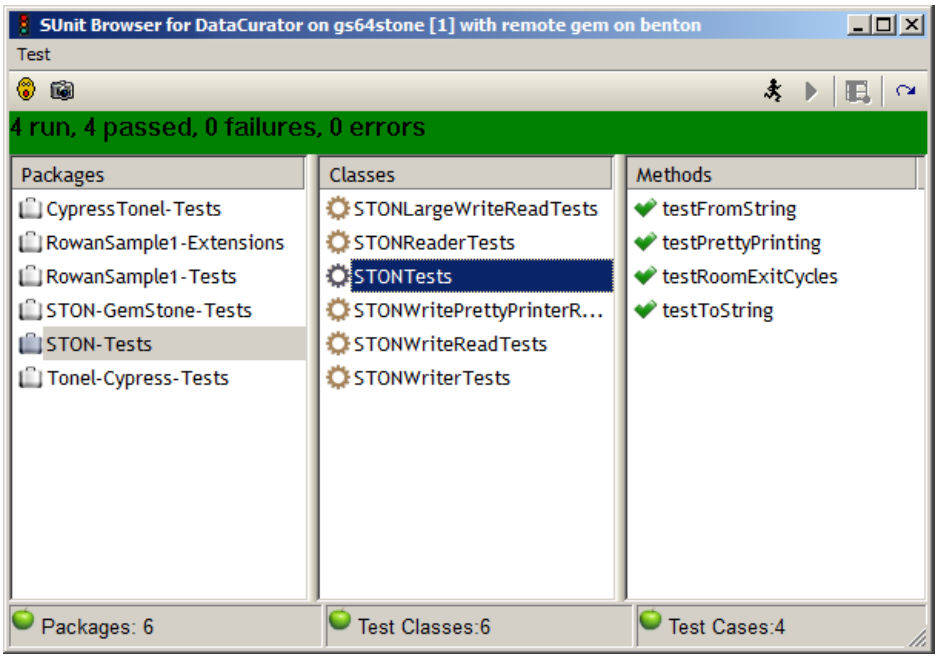


Class lookup is also available in the Method Source pane; this does require that the class name be selected.

SUnit Browser

The SUnit Browser allows you to see and run all tests that are visible to the user, anywhere within the image.

Figure 2.11 SUnit Browser

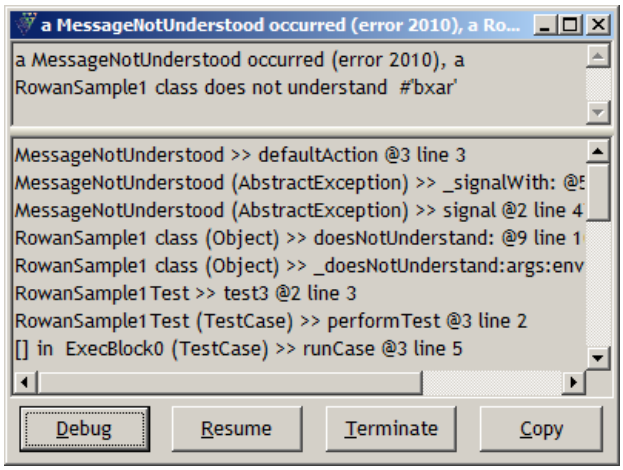


2.3 Debugger

The Debugger allows you both to examine object state in case of an unexpected error, and to step through code after setting a breakpoint.

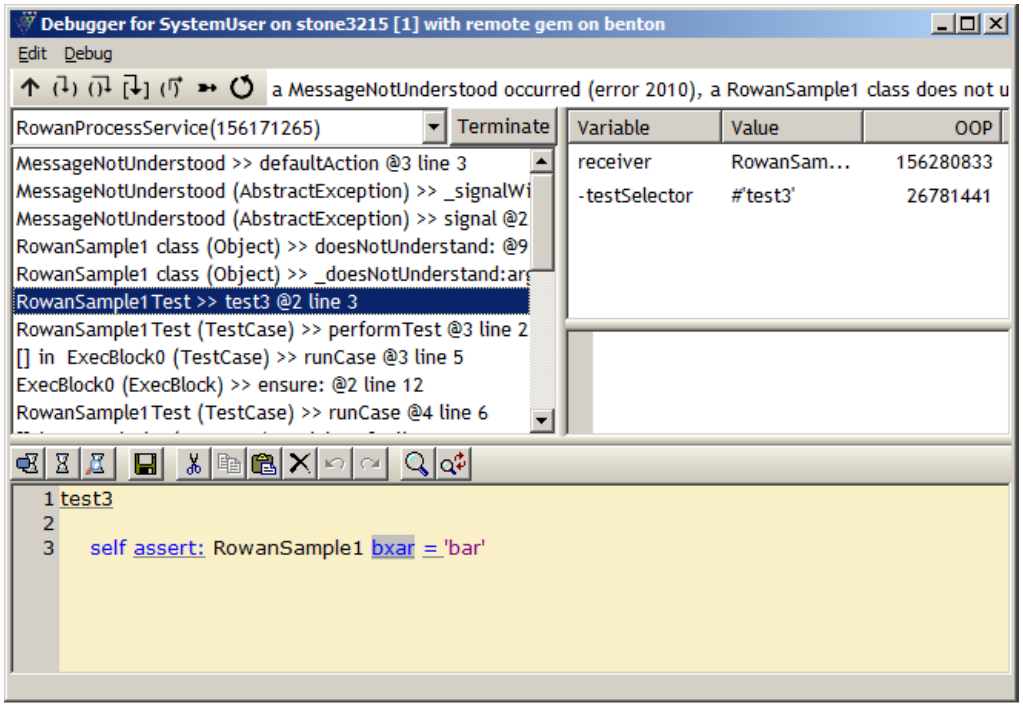
If an error occurs in the code, a **walkback window** is opened, which allows you to open the debugger, continue, terminate or copy the stack.

Figure 2.12 Walkback



The **Debug** button opens the debugger.

Figure 2.13 Debugger on code error



The debugger provides a subset of basic Smalltalk debugger operations as supported by the GemStone server. You may step into, step over, and a small number of similar operations.

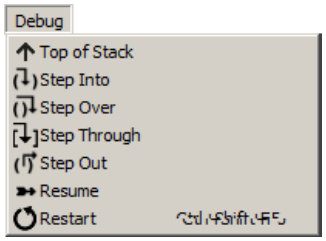
The upper right pane provides information on the variables in the selected context. The receiver is first; method arguments and temporaries are below in the same list, and you may need to scroll to see them. These values are not modifiable.

You may save method changes in the debugger, which will trim the stack.

Stepping through code

The debugger supports the following code options, available on the icon bar as well as the Debug menu:

Figure 2.14 Debug menu and toolbar options



The debugger supports the following code options

Step Into
Step, stepping into a method send or block

Step Over
Step, stepping over a method send or block

Step Out
Step out of a block or method send

Step Through
Step into a block, stepping through the infrastructure

Resume
Resume code execution

Restart
Restart code execution

Breakpoints

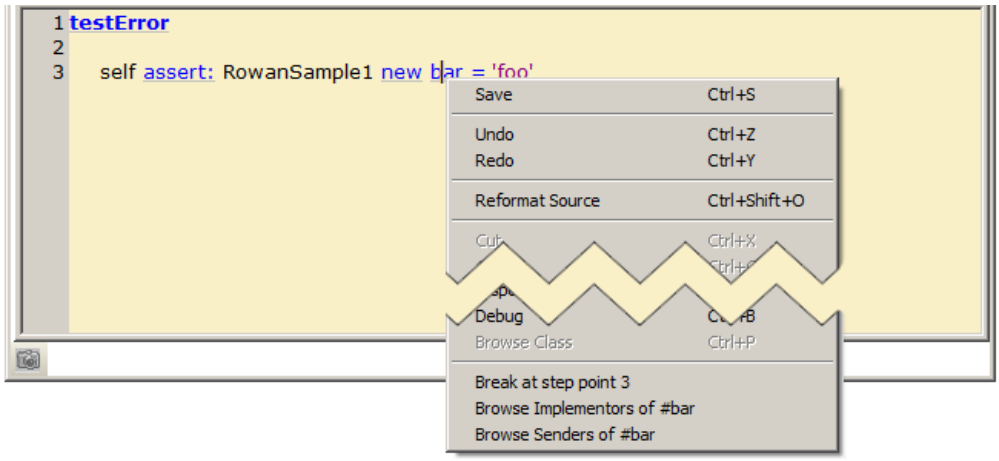
You can set breakpoints in your source code, which allows a debugger to open on the specific code you wish to examine.

To set a breakpoint, use the popup menu with the cursor over a selector or operation that is a step point; operations with step points have a gray underline (notice this underline under `assert:`, `new`, `bar`, and `=`, in the method below).

When popped up over a step point, the popup menu includes the option to set a break at that step point within the method.

For multi-keyword methods, you must have the cursor over the first keyword in the selector.

Figure 2.15 Setting breakpoint

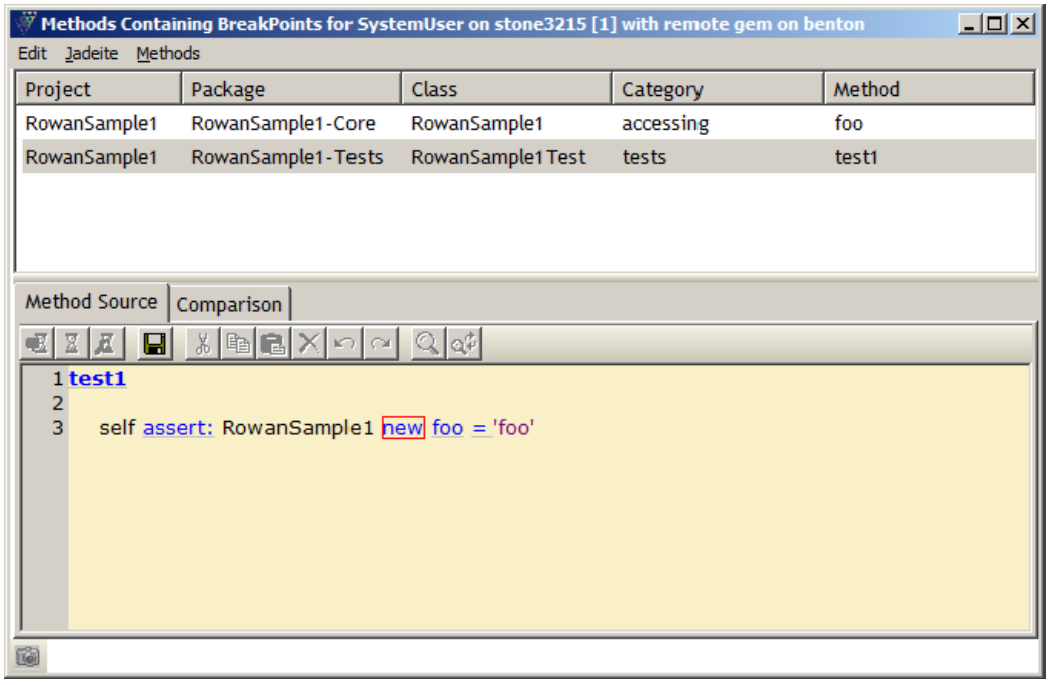


Breakpoints in method source code are displayed enclosed in a red box.

When code execution encounters a breakpoint, the debugger open.

Breakpoint Browser

Figure 2.16 Breakpoint Browser



The Breakpoint Browser allows you to examine all breakpoints in the system, and clear breakpoints for each method. You can open the Breakpoint Browser from the Project Browser menu item **Method > Browse Breakpoints**, or the Console menu item **Browse > Browse Breakpoints**.

The Breakpoint Browser is similar to a Method List Browser, but the menu bar **Method** drop down menu includes the additional menu item **Clear All Breakpoints.**

Disabling Breakpoints

Breakpoints can also be disabled system wide, using the new Console menu item **Jadeite > Breakpoints Enabled**.

When breakpoints are disabled, the normal red box outline becomes gray. When breakpoints are disabled, breakpoints can be added and removed, but will not halt execution. You can re-enable breakpoints at any time using the same menu item.

Breakpoints enabled is true by default, but can be defined in the preferences file, using a line such as:

```
breakpointsEnabled: false
```

2.4 Code Caveats

Non-Rowanized code

The Jadeite browsers display and allow you to work with both rowanized code that is in a package and project; and non-rowanized code. The GemStone kernel is not rowanized, while all application code is normally in Rowan projects. Be careful to avoid creating any non-rowanized application code. Non-rowanized application code isn't visibly different, but is not written to the git repository and is likely to result in errors.

- Do **not** use class or method creation or manipulation methods such as `compileAccessorMethodsFor:`, that do not have the `rw` prefix, to edit application code.
- Do not login using GBS or another tool to edit application code; you may browse code, but do **not** commit changes.
- The GemStone kernel code is not rowanized, and you should not modify this code (as SystemUser) using Jadeite.

Adding or modifying extension methods on GemStone kernel classes is supported, but there are special considerations for packaging due to Rowan's requirements for extension method's and package's SymbolDictionaries.

Aborts and the Transient Symbol Dictionary

When a project is loaded into a SymbolDictionary that was not previously defined, it is added to the transient symbol dictionary. Abort does not update the state of a transient symbol dictionary. If you make changes such as adding or removing classes, and perform an abort or attempt to reload the project, the state of the project becomes inconsistent.

When loading a project that defines a new SymbolDictionary, it is recommended that you commit your work, logout and login again.

To remove classes, use the class menu item **Remove** rather than abort.

Adding Packages to Project

A Package can be added to a Project using the Project Browser **Add Package...** menu item.

Adding a package **requires** editing the configuration file that defines the Project. See the *Rowan for GemStone User's Guide* for information on Project Configuration files. Otherwise, the new package will not be loaded; and a load operation from the Git repository into the image will remove the package.

Chapter

3 Recommended Workflow

Git is a powerful code management tool with many features. Without some care, however, unintended situations can arise that may require some effort to resolve.

To ensure that work flows smoothly, it is strongly recommended that all developers follow a disciplined workflow that takes into consideration all of the following:

- The need for individual developers to save their in-progress work.
- The need for multiple individuals to merge their work on shared code.
- The need to keep both a main, trunk development line and branch development lines in a usable state.

GemTalk recommends the following workflow for use with Jadeite and Rowan for GemStone.

Getting Started

Each developer should have their own local clone (copy) of the Git repository, and have their own branch in which to make code changes.

This process is described under "Browsing and Code Development" on page 21.

Code development

Code development using Jadeite is like any Smalltalk development. You may add, remove, and edit classes and methods using the Project Browser, method browsers, or debugger.

Keep in mind the relationship between package names and method category (protocol) names beginning with *, and class category names.

Each class is created in a package, and the `category:` keyword in the class creation template must match the name of a loaded Package.

Note that adding a new Package requires an additional step, editing the Project configuration, to make sure it is persistent.

You should commit your work frequently to your GemStone repository to be sure it is saved in case the session has an error or is lost. The GemStone commit commits to the local GemStone repository, and is entirely different than a Git commit.

Periodically, you may wish to commit to Git, using the Projects Browser **Commit...** menu item. This writes your code changes to the local Git repository, and does a Git commit. It does not share your work outside of your local Git repository.

Sharing work

When the code is working, you are ready to merge it into the shared repository. This involves make sure that is will work with any other recent changes before making it available to everyone.

This is a two-step process; first you will perform the merge in your own branch, which allows you to test your changes, and verify that your new code works with any other recent updates. Then, you will merge your branch into the master branch.

All this work is done in the local Git repository. Push and pull operations move the code changes between the local Git repository and the shared Git repository.

1. Checkout the master (this switches you to the master branch). This can be done using Jadeite in the Projects Browser **Checkout...** menu item, or using the Git command line, e.g.

   ```
   git checkout master
   ```

2. Pull the master branch from the shared Git repository. This updates your local Git repository with the most recent shared master code. This can be done in Jadeite using the Projects Browser **Pull from Git...** menu item, or using the Git command line, e.g.

   ```
   git pull master
   ```

3. Checkout your individual branch. This can be done in Jadeite using the Projects Browser **Checkout...** menu item, or using the Git command line, e.g.

   ```
   git checkout individual
   ```

4. Merge the master branch into your individual branch. If there are conflicts, the conflicts will need to be resolved before you proceed. This can be done using the Git command line, e.g.

   ```
   git merge master
   ```

5. Push the individual branch to the shared Git repository. This can be done in Jadeite using the Projects Browser **Push to Git...** menu item, or using Git command line, e.g.

   ```
   git push individual
   ```

6. Test the changes, to verify that the code runs as intended with the merged changes.

7. Once the merged changes are verified, checkout the master branch. This can be done in Jadeite using the Projects Browser **Checkout...** menu item, or using the Git command line, e.g.

   ```
   git checkout master
   ```

8. Merge the individual branch into the master. This can be done using the Git command line, e.g.

   ```
   git merge individual
   ```

9. Push the master branch to the shared Git repository. This can be done in Jadeite using the Projects Browser **Push to Git...** menu item, or using the Git command line, e.g.

   ```
   git push master
   ```

Figure 3.1 Code Sharing Git activity