GemStone®

GemBuilder™ for Java™ Tools

Version 2.2

October 2006

GemStone Systems, Inc.
1260 NW Waterhouse Avenue, Suite 200
Beaverton, OR 97006

Preface

About This Documentation

This documentation describes GemBuilder for Java® for Java™ 2.2 Tools, an integrated environment for programming GemStone® Smalltalk applications. These tools consist of the following:

- A *GemStone Browser* for examining, creating, and modifying GemStone classes and methods.
- A *Workspace* for running small pieces of GemStone Smalltalk.
- An *Inspector* for examining and modifying the state of GemStone objects.
- A *Debugger* for examining and modifying an execution stack created by GemStone Smalltalk execution.
- A *Class Version Browser* for examining a class history, inspecting instances, migrating instances, deleting versions, and moving versions to another class history.
- A *Breakpoint Browser* for enabling, disabling, and removing breakpoints in methods, as a debugging aid examining a class history, inspecting instances, migrating instances, deleting versions, and moving versions to another class history.

GemBuilder for Java tools may be used against any GemStone/S server product, including GemStone/S, the 32-bit server product, or GemStone/S 64 Bit versions 1.x or 2.x. While the server products are similar, some features differ. Gembuilder for Java behavior that depends on server behavior may therefore vary depending on which server product and version you are running with. In this documentation, the term “GemStone” is used when behavior is common; server specific differences are noted when necessary.

Assumptions

To make use of the information in this documentation, you need to be familiar with the GemStone server and with the GemStone Smalltalk programming language as described in the *GemStone/S Programming Guide*. That book explains the basic concepts behind the language and describes the most important GemStone kernel classes.

This documentation assumes that the GemStone system has been correctly installed on your host computer as described in the GemStone System Administrator’s Guide and that your system meets the requirements listed in your *GemBuilder for Java Installation Guide*.

How This Manual is Organized

Connecting to the GemStone Server explains how to communicate with the GemStone object server by initiating a GemStone session.

The Launcher explain how to use the Launcher tool to access the various tools.

The Browsers explains how to use the GemStone browsers and tools to create classes and methods in the GemStone server.

Related Tools describes how to use the GemStone workspace, inspectors, and the debugging tool with your GemStone Smalltalk code.

Working with GemStone Classes and Methods explains what you need to know in order to define new GemStone classes and add them to the system.

Other Useful Documents

- The *GemBuilder for Java Programming Guide*, which is provided in .pdf form with your GemBuilder for Java installation.
- The *Programming Guide* for GemStone/S or GemStone/S 64 Bit describes the GemStone System and the GemStone Smalltalk language.

- If you will be acting as a system administrator, or developing software for someone else who must play this role, you should read the *System Administration Guide* for GemStone/S or GemStone/S 64 Bit.

Technical Support

GemStone provides several sources for product information and support. The product-specific manuals and online help provide extensive documentation, and should always be your first source of information. GemStone Technical Support engineers will refer you to these documents when applicable.

GemStone Web Site: http://support.gemstone.com

GemStone's Technical Support website provides a variety of resources to help you use GemStone products. Use of this site requires an account, but registration is free of charge. To get an account, just complete the Registration Form, found in the same location. You'll be able to access the site as soon as you submit the web form.

The following types of information are provided at this web site:

Help Request allows designated support contacts to submit new requests for technical assistance and to review or update previous requests.

Documentation for GemBuilder for Java is provided in PDF format. This is the same documentation that is included with your GemBuilder for Java product.

Release Notes and **Install Guides** for your product software are provided in PDF format in the Documentation section.

Downloads and **Patches** provide code fixes and enhancements that have been developed after product release. Most code fixes and enhancements listed on the GemStone Web site are available for direct downloading.

Bugnotes, in the Learning Center section, identify performance issues or error conditions that you may encounter when using a GemStone product. A bugnote describes the cause of the condition, and, when possible, provides an alternative means of accomplishing the task. In addition, bugnotes identify whether or not a fix is available, either by upgrading to another version of the product, or by applying a patch. Bugnotes are updated regularly.

TechTips, also in the Learning Center section, provide information and instructions for topics that usually relate to more effective or efficient use of GemStone products. Some Tips may contain code that can be downloaded for use at your site.

Community Links provide customer forums for discussion of GemStone product issues.

Technical information on the GemStone Web site is reviewed and updated regularly. We recommend that you check this site on a regular basis to obtain the latest technical information for GemStone products. We also welcome suggestions and ideas for improving and expanding our site to better serve you.

You may need to contact Technical Support directly for the following reasons:

- Your technical question is not answered in the documentation.
- You receive an error message that directs you to contact GemStone Technical Support.
- You want to report a bug.
- You want to submit a feature request.

Questions concerning product availability, pricing, keyfiles, or future features should be directed to your GemStone account manager.

When contacting GemStone Technical Support, please be prepared to provide the following information:

- Your name, company name, and GemStone license number
- The GemStone product and version you are using
- The hardware platform and operating system you are using
- A description of the problem or request
- Exact error message(s) received, if any

Your GemStone support agreement may identify specific individuals who are responsible for submitting all support requests to GemStone. If so, please submit your information through those individuals. All responses will be sent to authorized contacts only.

For non-emergency requests, the support website is the preferred way to contact Technical Support. Only designated support contacts may submit help requests via the support website. If you are a designated support contact for your company, or the designated contacts have changed, please contact us to update the appropriate user accounts.

Email: support@gemstone.com

Telephone: (800) 243-4772 or (503) 533-3503

Requests for technical assistance may also be submitted by email or by telephone. We recommend you use telephone contact only for more serious requests that require immediate evaluation, such as a production system that is non-operational. In these cases, please also submit your request via the web or email, including pertinent details such error messages and relevant log files.

If you are reporting an emergency by telephone, select the option to transfer your call to the technical support administrator, who will take down your customer information and immediately contact an engineer.

Non-emergency requests received by telephone will be placed in the normal support queue for evaluation and response.

24x7 Emergency Technical Support

GemStone offers, at an additional charge, 24x7 emergency technical support. This support entitles customers to contact us 24 hours a day, 7 days a week, 365 days a year, if they encounter problems that cause their production application to go down, or that have the potential to bring their production application down. For more details, contact your GemStone account manager.

Training and Consulting

Consulting and training for all GemStone products are available through GemStone's Professional Services organization.

- Training courses are offered periodically at GemStone's offices in Beaverton, Oregon, or you can arrange for onsite training at your desired location.
- Customized consulting services can help you make the best use of GemStone products in your business environment.

Contact your GemStone account representative for more details or to obtain consulting services.

Contents

Chapter 3. The Browsers

Chapter 4. Related Tools

Chapter 5. Working with GemStone Classes and Methods

Chapter

1

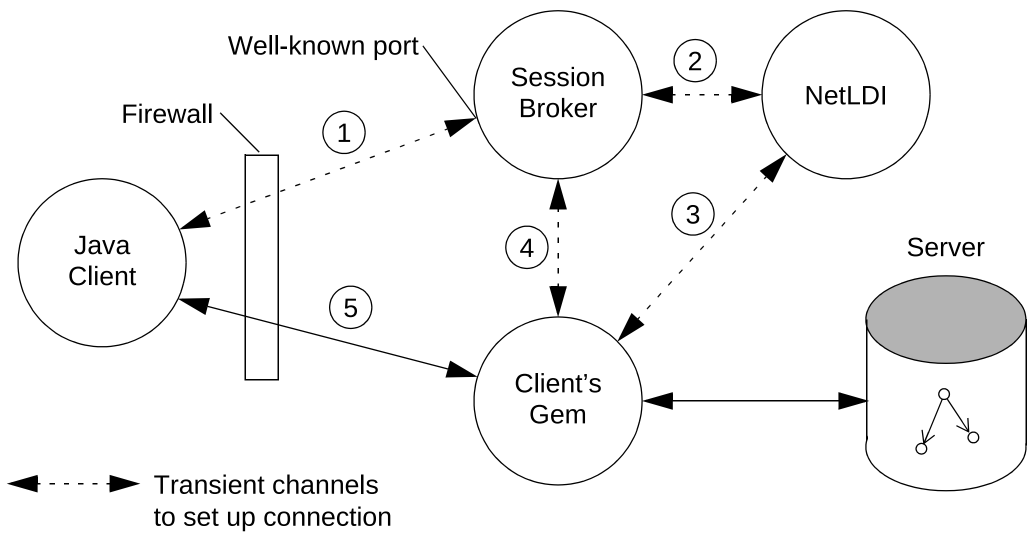
Connecting to the GemStone Server

Overview

The GemBuilder for Java Tools are Java applications that make use of the underlying GemBuilder for Java application programming interface. You can use these tools by themselves, or you can integrate them into popular Java development environments like Microsoft Visual J++, Sun's Java Workshop, and Symantec Café by following instructions in our *Installation Guide*.

All interaction with the GemStone server takes place through a Gem process that is created to serve your Tools session. An intermediary called the *Session Broker* is responsible for starting the Gem process (through a NetLDI network server) and for placing the Gem and the Tools in communication with each other.

Figure 1.1 Session Broker Connecting a Client to the GemStone Repository



A GemStone administrator starts the Session Broker from the command line or from a Topaz session using a chosen TCP / IP port number. During login, the Tools use the port number and the name of the GemStone server to contact the Session Broker and have it start the Gem session process.

For information on configuration and starting the Session Broker, see the *GemBuilder for Java Programming Guide*

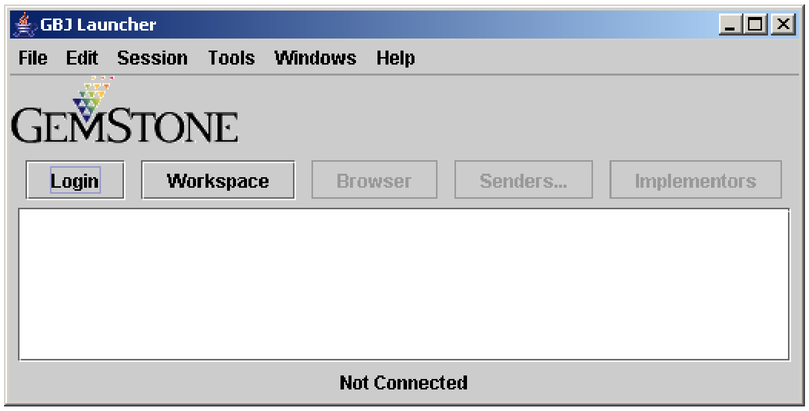
In order to log in from the Tools, you must have at hand at least the following information:

- the name of the GemStone server, such as gemserver61
- the name of the machine on which the Session Broker is running
- the port number at which the tools launcher is to contact the Session Broker, such as 9090 (the default)
- the GemStone user ID under which you will log in, and the corresponding password.

Launching the Tools

The GemStone Launcher window lets you log in to a GemStone server, which gives you access to the GemStone Smalltalk compiler and the GemBuilder for Java Tools. Initially, only the **Login...** and **Workspace** buttons are enabled. The lower pane, the text pane, serves as a transcript for system messages.

Figure 1.2 GemStone Tools Launcher



You can run the Launcher as a stand-alone Java application within a Web browser.

To Start a Standalone Launcher

To start the Launcher as a standalone application, invoke the Java runtime system from an operating system command prompt. Your CLASSPATH environment variable must include the location of `gbj22.jar`, as described in the documentation for your Java system and our *Installation Guide.* For example:

```
prompt> java com.gemstone.tools.GbjLauncher
```

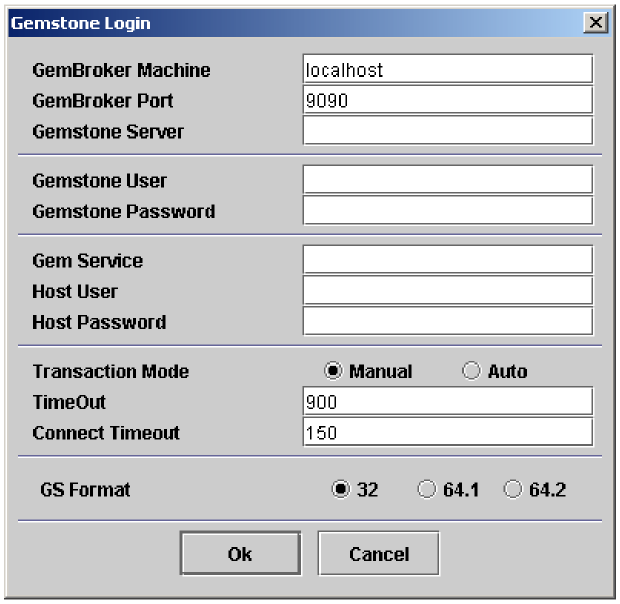
or

```
prompt> java -jar gbj22.jar
```

Logging in to GemStone

After the Launcher appears, the next step ordinarily is to log in to a GemStone server. Click the **Login...** button in the Launcher, which opens a dialog in which you identify the GemStone server and provide the GemStone userId, password, and other relevant information (the client can provide this information programmatically). Use the mouse or Tab key (not Return) to move between fields.

Figure 1.3 Login Dialog



Gemstone Login

GemBroker Machine	localhost
GemBroker Port	9090
Gemstone Server	
Gemstone User	
Gemstone Password	
Gem Service	
Host User	
Host Password	
Transaction Mode	[✓ Manual] Auto
TimeOut	900
Connect Timeout	150
GS Format	[✓ 32] 64.1 64.2

NOTE:
After you log in, all tools you start will run under that GemStone session. To run tools under a different session, start the Launcher again.

Parameter	Explanation
GemBroker Machine	The name or IP address of the machine where the Session Broker is running (the default “localhost” may be used if the Session Broker is on your machine)
GemBroker Port	The number of the well-known port at which the Session Broker is listening: either 9090 or the number provided by the GemStone administrator who started the Session Broker.

GemStone Server	The name of the GemStone server, such as "gemserver61". This field can include the name of server's machine using Network Resource String (NRS) syntax; for instance, "!@handel!gemserver61". GemBuilder for Java must be installed on the server.
GemStone User	The userId under which you want to log in (must already exist in the server)
GemStone Password	The password for the GemStone userId
Gem Service (optional)	The name of the Gem service to be used. Initially, the default (not shown) is "gemnetobject"; users of the UNIX C Shell may prefer "gemnetobjcsh" (not available with GemStone / S 64 Bit). Because the Gem and Broker must be on the same machine, you should not specify a machine name as part of an NRS.
Host User	An operating system login name on the machine where the Session Broker is running. This user name and password are required to start a separate Gem process unless your GemStone installation uses another method of providing network authentication, such as a .netrc file or running NetLDIs in guest mode with a captive account.
Host Password	The operating system password for the Host User on the machine where the Session Broker is running
Transaction Mode	Choose **Automatic** if you are going to be committing changes to the server. Choose **Manual** if your primary activity will be reading the committed state; before making a change you want to commit, begin a transaction manually by choosing **Session > Begin** in the GemStone Browser. (For more information, see the *GemBuilder for Java Programming Guide* or the *GemStone/S Programming Guide*.)
Timeout	The length of time in seconds that the server will wait before it drops an idle session. If zero, the connection will not time out. The default is 900 seconds (15 minutes).

Connect Timeout	The length of time in seconds that Java client will wait for a connection request to be fulfilled by the Session Broker. The default is 30 seconds.
GS Format	The OOP format of the GemStone server. The default is **32**, for use with GemStone / S version 6.x. For GemStone / S 64 Bit version 1.x, use **64.1**; for GemStone / S 64 Bit version 2.x, use **64.2**.

For information about Network Resource Syntax and the NetLDI, see the *GemStone System Administration Guide*.

Logging out of GemStone

When your login succeeds, the button label is changed to **Logout**. Click this button to end your session. All browsers, inspectors, and debuggers are closed automatically, but the launcher and any workspaces remain open. The workspaces remain open so you can save text, but you can no longer run code.

Chapter

2

The Launcher

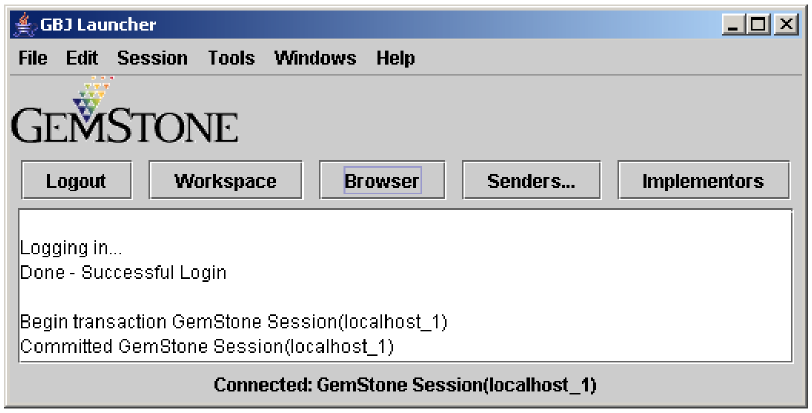
Overview

The Launcher not only lets you log in to a GemStone server, but also gives access to the other Tools:

- GemStone Browsers and specialized browsers derived from it
- Workspaces
- Inspectors

The Launcher contains a text pane that serves as both a session transcript and as a workspace. The transcript records significant events, such as logins, commits, and logouts. Clicking **Help** displays version information in the transcript.

Figure 2.1 Session Transcript in Launcher Text Pane



```
Logging in...
Done - Successful Login

Begin transaction GemStone Session(localhost_1)
Committed GemStone Session(localhost_1)
```

Connected: GemStone Session(localhost_1)

The Launcher Buttons

Five Launcher buttons provide convenient access to frequently used functions: Login / Logout, Browser, Workspace, Senders…, and Implementors… .

The **Login** button opens a dialog through which you enter session parameters and log in to the GemStone server, as described in ”Launching the Tools” on page 1-3. When the login attempt succeeds, the label on this button changes to **Logout**.

The **Browser** button opens a GemStone Browser through which you can examine, create, or modify Smalltalk classes and methods in the server. For further information, see ”The Browsers” on page 3-1. This button is enabled only when you are logged in to the server.

The **Workspace** button opens a text area in which you can evaluate GemStone Smalltalk expressions or edit text. This button is always enabled, but to evaluate expressions, you must be logged in to the server.

The **Senders…** button prompts for a method name, then opens a Method List Browser showing all visible server methods that send the specified message. For further information, see ”The Method Browser” on page 3-12. This button is enabled only when you are logged in to the server.

The **Senders…** and **Implementors…** buttons search the methods visible in the session's symbol list. For more information about how GemStone uses the symbol list for name resolution, see ”The Symbol List Pane” on page 3-2 and the *GemStone/S Programming Guide*.

You can use wild-card characters in the search dialog associated with **Senders…** and **Implementors…**. A “*” matches any number of characters, and “?” matches a single character.

The Launcher Menus

The Launcher menus provide commands for executing GemStone Smalltalk code and accessing the GemStone Tools.

The File Menu

The File menu contains these items:

Open...	Requests a file name, then copies the contents into the workspace (transcript)
Save As...	Requests a file name, then writes the contents of the workspace (transcript) into the specified file
File In…	Opens a file dialog allowing you to specify a source code file. Treats the selected file as being in GemStone file-in format (see ”Reading and Compiling a Saved File” on page 5-11) and files it in.
Settings	Opens a dialog that allows you to specify a font and font style; and allows you to set verbose client logging **On** or **Off** for the client and server. Changes affect the Launcher and any tools you open subsequently.
Exit	Kills the Tools application, closing all open browsers, inspectors, and so forth. The GemStone session is terminated.

The Edit Menu

The Edit menu contains these items:

Cut	Cuts the currently selected text from the text pane and puts it in the clipboard.
Copy	Copies the currently selected text from the text pane and puts it in the clipboard.
Paste	Pastes the text from the clipboard at the current selection position in the text pane.
Select All	Selects all of the text in the text pane
Do-It	Uses the GemStone compiler to evaluate and execute the currently selected text
Print-It	Performs a **Do-It**, sends the resulting object the message printString, then places the resulting string after the current text pane selection
Inspect-It	Performs a **Do-It**, and then opens a GemStone Inspector on the resulting object
File It In	Treats the selected text as being in GemStone file-in format (see ”Reading and Compiling a Saved File” on page 5-11) and files it in.

The Session Menu

The Session menu contains these items:

Login…	If there is no Launcher session, opens a login dialog and lets you log in to GemStone (same as clicking the **Login…** button)
Logout	Logs out the Launcher session, if there is one (same as clicking the **Logout** button)
Commit	Commits the Launcher session
Abort	Aborts (rolls back) the Launcher session
Begin	Begins a new transaction
Manual	Switches the Launcher session to manual transaction mode
Auto	Switches the Launcher session to automatic transaction mode

The Tools Menu

The Tools menu contains these items:

Browser	Opens a GemStone System Browser (same as clicking the **Browser** button)
Workspace	Opens a GemStone workspace in which you can execute Smalltalk expressions (same as clicking the **Workspace** button)
Senders	Requests a selector name, then opens a Method List Browser showing all senders of that message (same as clicking the **Senders…** button)
Implementors	Requests a selector name, then opens a Method List Browser showing all implementors of that message (same as clicking the **Implementors…** button)
References	Requests a symbol and then opens a Method List Browser showing all methods that refer to that symbol, including all references to associations with the specified symbol as the key.
Find Substring	Requests a string, and then opens a Method List Browser showing all methods that contain the specified string in their source code. The search is case-sensitive. If no such string is found, opens a notifier instead.
Browse Class	Requests a class name, then opens a Class Browser on the class
Browse Hierarchy	Requests a class name, then opens a Hierarchy Browser on the class
Breakpoints	Opens a Breakpoint Browser.

You can use wild-card characters in the search dialog associated with **Senders**, **Implementors**, **Browse Class**, and **Browse Hierarchy**. A “*” matches any number of characters, and “?” matches a single character.

The Windows Menu

The Windows menu contains these items:

Hide All	Hides all windows.
Show All	Shows all windows.
<name of each open window>	Each open window has an entry in the menu. You can show an individual window by clicking that window in the Windows menu.

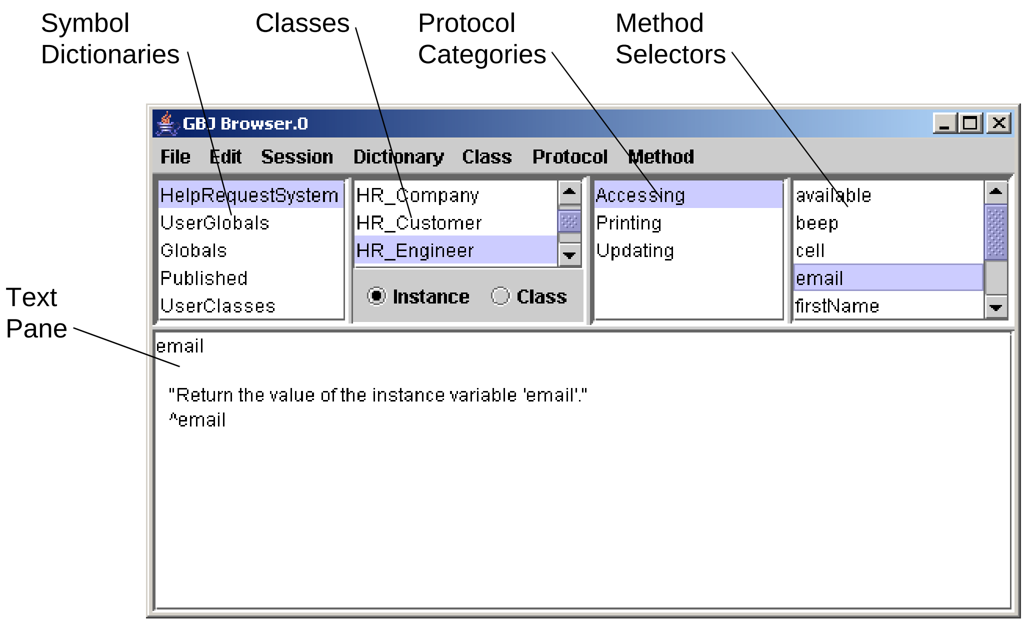
Chapter 3

The Browsers

The GemStone Browser

After you have successfully logged in to GemStone, you can open a GemStone Browser by choosing **Tools > Browser** or by clicking the **Browser** button in the Launcher. The GemStone Browser lets you create classes and methods in the server using GemStone Smalltalk.

Figure 3.1 GemStone Browser



```
email

 "Return the value of the instance variable 'email'."
 ^email
```

The Symbol List Pane

The upper left pane contains a list of symbol dictionaries. In GemStone, the named objects to which your programs refer (the classes and root or global objects) are listed in symbol dictionaries. This list facilitates finding objects efficiently as well as sharing them among users. All of the symbol dictionaries that you can access in your current session environment are listed in the GemStone Browser's Symbol List pane.

When you select a symbol dictionary, all classes defined in that dictionary appear in the pane to the right of the Symbol List pane. (Symbols other than classes can be viewed by opening an inspector on the symbol dictionary in question.)

For further information about symbol lists, see the *GemStone/S Programming Guide*.

The Classes Pane

When you select a class from the Classes pane, a list of its protocols appears in the Protocol pane adjacent on the right. In GemStone, methods are categorized by function, or *protocol*, to make them easier to browse.

The Protocol Pane

When you select a protocol, all the message selectors in that category appear in the Method Selectors pane at the far right.

You can switch the focus of the display between instance methods or class methods by using the radio buttons provided.

The Text Pane

When you select a method, its source code is displayed in the lower portion of the browser—the text pane. In this pane, you can edit and recompile the text of the displayed method, or execute fragments of GemStone Smalltalk code as in a workspace.

The Browser Menus

The File Menu

The following commands are available in the File menu:

Update	Updates the browser to match the current state of the Launcher's session. This action may change the contents of the browser because of changes made in other tools.
Close	Close the browser window.

The Edit Menu

The Edit menu contains these items:

Cut	Cuts the currently selected text from the text pane and puts it in the clipboard.
Copy	Copies the currently selected text from the text pane and puts it in the clipboard.
Paste	Pastes the text from the clipboard at the current selection position in the text pane.
Select All	Selects all of the text in the text pane
Do-It	Uses the GemStone compiler to evaluate the currently selected text

Print-It	Performs a **Do-It**, sends the resulting object the message printString, then places the resulting string after the current text pane selection
Inspect-It	Performs a **Do-It**, then opens a GemStone Inspector on the resulting object
Compile	Compiles the current text in the text pane. Use this choice to create a new class or method, or to modify existing definitions.
Cancel	Cancels any text entry done in the text pane, and returns the method definition, class definition, and so forth, to its unmodified form
Set Break	Sets a breakpoint at the current insertion point in the current method.

The Session Menu

The Session menu contains these items:

Commit	Commits the Launcher session
Abort	Aborts (rolls back) the Launcher session
Begin	Begins a new transaction

If an attempted commit fails, choose **Report** in the resulting dialog to open a GemStone Inspector through which you can determine the cause of the failure. The inspector is on the result of invoking System Class >> transactionConflicts in the server.

The Dictionary Menu

The Dictionary menu contains these items:

File Out As	Requests a file name, then creates a fileout (.gs file) containing the classes and methods in the selected symbol dictionary.
Inspect	Opens a GemStone Inspector on the currently selected symbol dictionary.
Add	Requests a new dictionary name, then adds a new symbol dictionary by that name to your symbol list.
Rename As	Requests a new dictionary name, then renames the selected symbol dictionary to that name.
Remove	After confirmation, removes the selected symbol dictionary from your symbol list. *WARNING: Do not try to remove the Globals dictionary. The Globals dictionary defines the GemStone kernel classes.*
Find Class	Requests a class name, then selects the matching symbol dictionary and class entry in the class list.

The Class Menu

The Class menu is active only when a class has been selected. The following commands are available from the Class menu. (”Defining a New Class” on page 5-4 explains how to define a new GemStone Smalltalk class to add to the currently selected symbol dictionary.)

File Out As...	Requests a file name, then creates a fileout (.gs file) containing the selected class and its methods
Browse > Class	Opens a Class Browser on the selected class
Browse > Hierarchy	Opens a Class Hierarchy Browser on the selected class.
Browse > Versions	Opens a Class Version Browser on the selected class. The browser shows how many versions of that class exist.
References To > Instance Variable	Opens a selection dialog allowing you to choose from a list of instance variables of the selected class. Choose one and click **OK** or press Return, and a Method List Browser opens showing all methods that refer to the selected instance variable.

References To > Class Variable	Opens a selection dialog allowing you to choose from a list of class variables of the selected class. Choose one and click **OK** or press Return, and a Method List Browser opens showing all methods that refer to the selected class variable.
References To > Class	Opens a Method List Browser showing all methods that refer to the selected class.

Show > Hierarchy	Lists the superclasses of the selected class and their instance variables. For example, if the current class is PositionableStream, the hierarchy list appears as follows:
	Object ()
	  Stream ()
	    PositionableStream (itsCollection position)
	      ReadStream ()
	      WriteStream ()
	The names in parentheses in this example show that PositionableStream has two named instance variables.Lists the superclasses of the selected class and their instance variables in a hierarchy. Names in parentheses show named instance variables.

Show > Definition	Displays the class definition of the selected class in the text pane.
Show > Comment	Displays the class comment of the currently selected GemStone-supplied class
Move to	Lets you choose another symbol dictionary, then moves the selected class to that dictionary.
Remove	After confirmation, removes the selected class from its associated symbol dictionary *CAUTION: To avoid inadvertently removing or modifying a GemStone kernel class, use the DataCurator account for all administrative tasks except those that require SystemUser privileges, such as upgrading the repository.*
New Template	Presents a class definition template in the text pane.

The Protocol Menu

To help you organize your work, GemStone Smalltalk methods that are functionally similar are grouped together into protocol categories with descriptive names, such as *initializing*, *accessing*, or *updating*. These categories are descriptive only; they do not affect the operation of GemStone Smalltalk in any way.

The following items are available in the Protocol menu. (Until you've selected a method category, Add is the only menu item enabled.)

Add	Requests a protocol name, then adds the protocol to the selected class.
Rename As	Requests a protocol name, then renames the selected protocol to that name.
Remove	Removes the selected protocol (after confirmation, if it contains any methods).
Find Method	Provides a list of all method names in the selected class, then updates the protocol and method panes to show the selected protocol and method.

The Method Menu

The Method menu allows you to file out a single method, reorganize methods into different categories, and remove a method. Items in the Method menu are available only when a method has been selected.

File Out As	Requests a file name, then creates a fileout (.gs file) containing the selected method.
Browse Method	Opens a GemStone Method Browser on the selected method.
Senders	Opens a GemStone Method List Browser showing all senders of the selected method (same as clicking the **Senders…** button in the Launcher and entering the name of the method).
Implementors	Opens a GemStone Method List Browser showing all implementors of the selected method (same as clicking the **Implementors…** button in the Launcher and entering the name of the method).

Messages	After you choose one of the messages sent by the selected method, opens a GemStone Method List Browser showing all implementors.
Move to	Requests the name of a protocol to which the selected method should be moved, then moves the method. If no protocol exists by the name supplied, this menu item adds the protocol and then does the move.
Remove	After confirmation, removes the selected method.
View > Class Only	Displays protocols and methods for the selected class only. The selected view persists when you select a new class.
View > Up To Root	Displays protocols and methods for the selected class and all superclasses up to, but not including, the class at the root of the hierarchy (typically Object). The selected view persists when you select a new class.
View > Including Root	Displays protocols and methods for the selected class and all superclasses, including the class at the root of the hierarchy (typically Object). The selected view persists when you select a new class.
New Template	Presents a new method template in the text pane.

Specialized Browsers

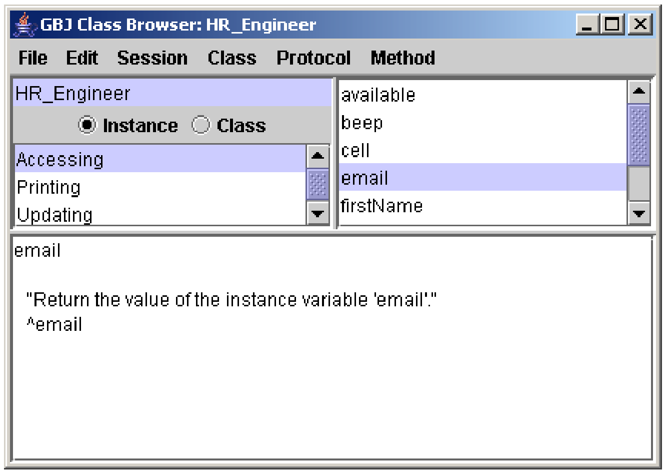
GemBuilder for Java Tools implement several specialized browsers based on the GemStone Browser: the Class Browser, Hierarchy Browser, Class Version Browser, Method Browser, Method List Browser, and Breakpoint Browser.

The Class Browser

The GemStone Class Browser displays a single GemStone class. The menu items supported by the browser are the same as those supported by the System Browser except the Dictionary menu is not available. Open this browser by choosing **Class > Browse > Class** in the GemStone Browser or by choosing **Tools > Browse Class** in the GemStone Launcher.

You can use wild-card characters in searches, with “*” matching any number of characters, and “?” matching a single character.

Figure 3.2 Class Browser



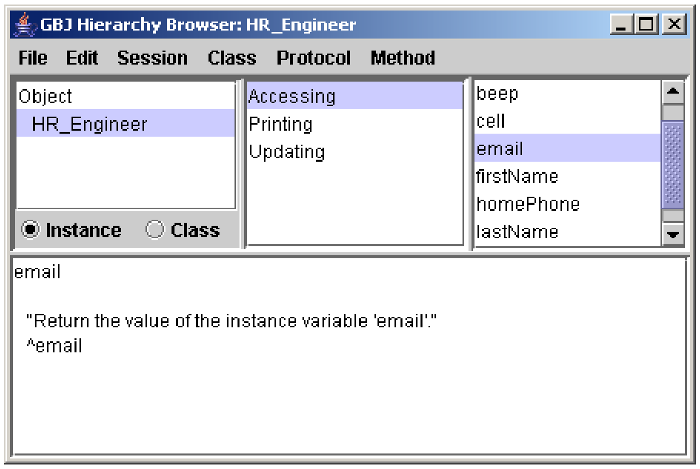
```
email

 "Return the value of the instance variable 'email'."
 ^email
```

The Hierarchy Browser

The GemStone Hierarchy Browser displays the superclass chain of a target class plus all subclasses of the target class. For example, choosing **Class > Browse > Hierarchy** when class HR_Engineer is selected allows you to browse the hierarchy for that class, which happens to be a direct subclass of Object. The menu items supported by the browser are the same as those supported by the System Browser, except the Dictionary menu is not available.

You can use wild-card characters in searches, with "*" matching any number of characters, and "?" matching a single character.

Figure 3.3 Class Hierarchy Browser

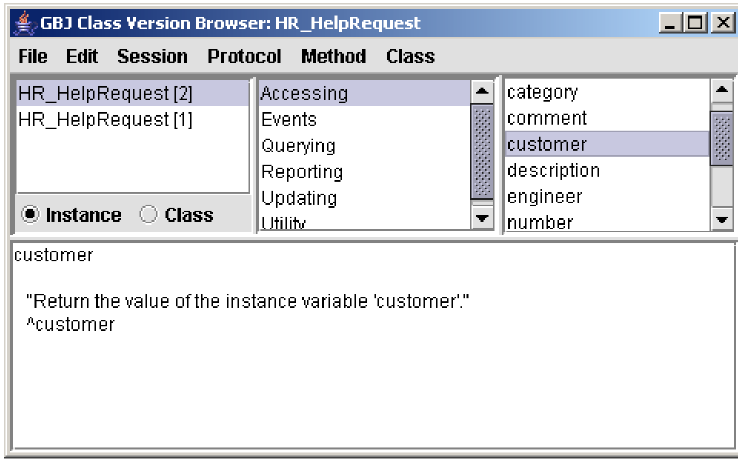


```
email

 "Return the value of the instance variable 'email'."
 ^email
```

The Class Version Browser

The GemStone Class Version Browser displays the versions of a class in the class history of the target class. For example, doing **Class > Browse Versions** when class HR_HelpRequest is selected in the GemStone Browser shows two versions exist. You can select either version in the left pane. The browser displays the definition of the selected version in the text pane.

Figure 3.4 Class Version Browser



```
customer

  "Return the value of the instance variable 'customer'."
  ^customer
```

The menu items supported by the Class Version Browser are generally the same as those supported by the System Browser. The Class menu has been extended, and several of the menu items have slightly different semantics. The additional and altered items in the Class menu are these:

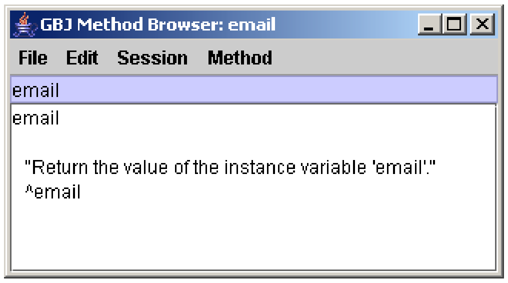
Class > Move to	Requests the name of a target class, then moves the selected class version to the class history of the target class.
Class > Remove	After confirmation, removes the selected class version from its class history.
Class > Inspect Instances	Opens a GemStone Inspector on all instances of the currently selected class version. Because this operation can be lengthy, you are asked to confirm your intent.

Class > Migrate Instances	After you choose a target class version, migrates all instances of the selected version to the target version.

The Method Browser

The Method Browser displays an individual method. For example, invoking the Method browser on `HR_Engineer > email` shows its definition.

Figure 3.5 Method Browser



```
email

 "Return the value of the instance variable 'email'."
 ^email
```

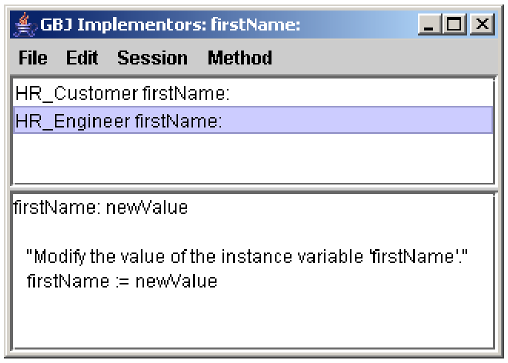
The supported menu items are the same as those for the other browsers except the Dictionary, Class and Protocol menus are not available.

The Method List Browser

The Method List Browser displays a list of methods defined in various classes. The browser is used to display senders and implementors of various messages. For example, invoking the menu item Implementors on firstName*:* lets you browse each implementation of that message. The same list can be obtained by clicking the Implementors... button in the GemStone Launcher.

You can use wild-card characters in searches, with “*” matching any number of characters, and “?” matching a single character.

Figure 3.6 Implementors of a Particular Method



HR_Customer firstName:
HR_Engineer firstName:

```
firstName: newValue

  "Modify the value of the instance variable 'firstName'."
  firstName := newValue
```

The Breakpoint Browser

The Breakpoint Browser displays a list of methods for which breakpoints have been specified. Open it using the **Tools > Breakpoints** menu item in the GemStone Launcher. You can then select a breakpoint and enable, disable, or remove it.

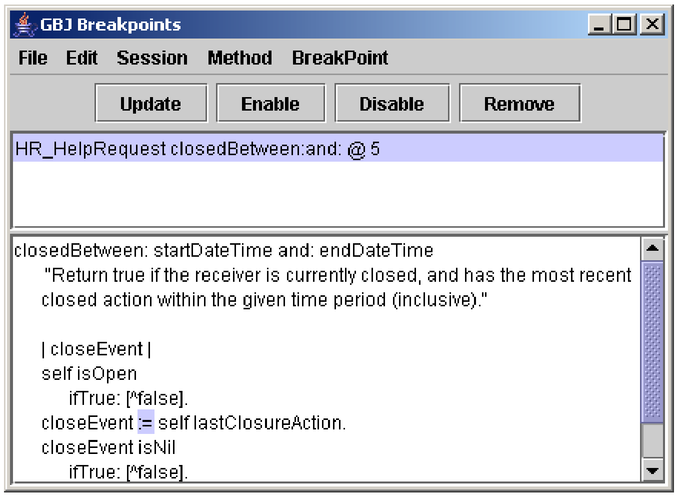
To set a breakpoint:

1. Open any tool that allows you to browse method source code.
2. Place the insertion point before the point at which you wish execution to halt.
3. Select **Edit > Set Break**.

When a breakpoint is encountered, a notifier allows you to open a debugger. See ”The Debugger” on page 4-6.

The Breakpoint Browser appears as shown below:

Figure 3.7 Viewing Breakpoints


GBJ Breakpoints
File Edit Session Method BreakPoint
Update Enable Disable Remove

HR_HelpRequest closedBetween:and: @ 5

```
closedBetween: startDateTime and: endDateTime
    "Return true if the receiver is currently closed, and has the most recent
    closed action within the given time period (inclusive)."

    | closeEvent |
    self isOpen
        ifTrue: [^false].
    closeEvent := self lastClosureAction.
    closeEvent isNil
        ifTrue: [^false].
```

The top pane contains a list of all breakpoints set in GemStone Smalltalk code. The bottom pane shows the source for the selected breakpoint, highlighting the message-send at which execution will stop when the breakpoint is encountered.

Breakpoint Browser Buttons

Four buttons provide convenient access to the main functions of the Breakpoint Browser.

The **Update** button updates the Breakpoint Browser in case breakpoints have been set in another tool since the browser was opened.

The **Remove** button removes the selected breakpoint.

The **Enable** button enables the selected breakpoint, causing execution to stop when it is reached.

The **Disable** button disables the selected breakpoint without removing it.

Breakpoint Menu

The Breakpoint Browser offers one new menu, the Breakpoint menu. It contains these items:

Remove Break	Removes the currently selected breakpoint from the browser and the system. This is the same as clicking the **Remove** button.
Enable Break	Enables the currently selected breakpoint, causing execution to halt when it is reached. This is the same as clicking the **Enable** button.
Disable Break	Disables the currently selected breakpoint, causing execution to continue uninterrupted when it is reached, but leaving the breakpoint in both the system and the browser for possible future use. This is the same as clicking the **Disable** button.
Remove All Breaks	Removes all breakpoints from the browser and the system.
Enable All Breaks	Enables all breakpoints, causing execution to halt when they are reached.
Disable All Breaks	Disables all breakpoints, causing execution to continue uninterrupted when they are reached, but leaving them in both the system and the browser for possible future use.

Chapter 4 Related Tools

Overview

Three additional tools provide assistance in working with GemStone Smalltalk:

- *Workspaces* are text areas in which Smalltalk expressions can be evaluated. You can also use a workspace to file in code previously saved from GemStone.
- *Inspectors* let you examine GemStone objects, including the result of evaluating an expression in a workspace.
- The *Debugger* lets you inspect the state of Smalltalk execution at the time of an error.

In addition, certain menus and keyboard shortcuts are common to all the tools.

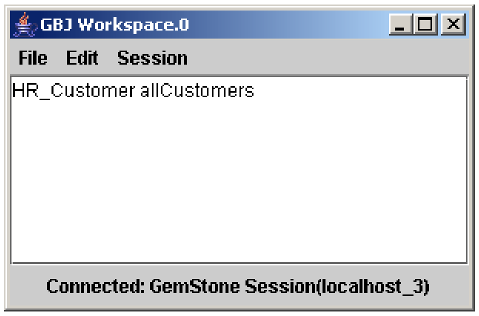
The Workspace

The GemStone Workspace provides a text area in which expressions can be evaluated. For example, you can evaluate the expression

```
HR_Customer allCustomers
```

by highlighting the expression and then choosing **Edit > Print-It** or a similar menu item.

Figure 4.1 A Workspace



HR_Customer allCustomers

Connected: GemStone Session(localhost_3)

The menu items available in the Workspace menu are similar to those in the Browser and Launcher:

File > Open	Requests a file name, then copies the contents into the workspace.
File > Save as	Requests a file name, then writes the contents of the workspace into the specified file.
File > File In	Opens a file dialog allowing you to specify a source code file. Treats the selected file as being in GemStone file-in format (see ”Reading and Compiling a Saved File” on page 5-11) and files it in.
File > Close	Closes the workspace.
Edit > Cut	Cuts the currently selected text from the text pane and puts it in the clipboard.
Edit > Copy	Copies the currently selected text from the text pane and puts it in the clipboard.
Edit > Paste	Pastes the text from the clipboard at the currently selected position in the text pane.

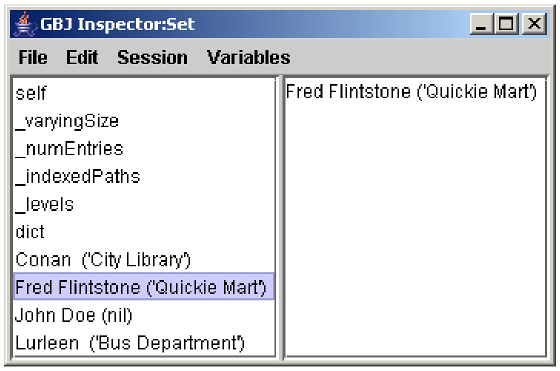
Edit > Select All	Selects all of the text in the text pane.
Edit > Do-It	Uses the GemStone compiler to evaluate the currently selected text.
Edit > Print-It	Performs a **Do-It**, sends the resulting object the message printString, and then places the resulting string after the current text pane selection.
Edit > Inspect-It	Performs a **Do-It**, then opens a GemStone Inspector on the resulting object.
Edit > File-It In	Treats the selected text as though it is in GemStone file-in format, and tries to file it in to the GemStone server.
Session > Commit	Commits the Launcher's session.
Session > Abort	Aborts the Launcher's session.
Session > Begin	Begins a new transaction.

The Inspector

When you select a GemStone Smalltalk expression and execute **Inspect-It** from the **Edit** menu, a GemStone Inspector opens. This inspector contains two panes, which enable you to examine and modify the values of instance variables in the GemStone server object returned as the result of the expression.

The left pane lists `self`, which refers to the inspected object, together with the names or indexes of the object's instance variables. (If the object is an instance of a Dictionary, the left pane lists the keys). When you select an item in the left pane, the right pane shows the value of that item. For example, invoking an Inspector on the expression `HR_Customer allCustomers` produces this inspector:

Figure 4.2 An Inspector



self
_varyingSize
_numEntries
_indexedPaths
_levels
dict
Conan ('City Library')
Fred Flintstone ('Quickie Mart')
John Doe (nil)
Lurleen ('Bus Department')

Fred Flintstone ('Quickie Mart')

You can use the right pane to execute GemStone Smalltalk expressions that contain names appearing in the left pane. For example, if the inspected object had an instance variable named *count* whose value was 5, then executing the expression `count * 5` in the right pane returns the value 25.

To modify the value of an item shown in the left pane, select the item, then type a GemStone Smalltalk expression in the right pane. The choose **File > Save**. The value of the selected object in the left pane becomes the value returned by the expression

NOTE:
You cannot modify the value of `self` *in an inspector in this way*

The menu items supported by the GemStone Inspector are shown below.

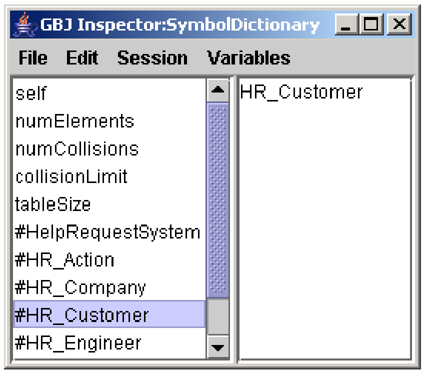
File > Update	Updates the inspector to match the current state of the Launcher's session.
File > Close	Closes the inspector window.
Edit > Cut	Cuts the currently selected text from the text pane and puts it in the clipboard.
Edit > Copy	Copies the currently selected text from the text pane and puts it in the clipboard.
Edit > Paste	Pastes the text from the clipboard at the current selection position in the text pane.
Edit > Select All	Selects all of the text in the text pane.

Edit > Do-It	Uses the GemStone compiler to evaluate the currently selected text.
Edit > Print-It	Performs a **Do-It**, sends the resulting object the message printString, and then places the resulting string after the current text pane selection.
Edit > Inspect-It	Performs a **Do-It**, then opens a GemStone Inspector on the resulting object.
Edit > Compile	Compiles the current text in the text pane. The text in the text pane is evaluated, and the resulting object is assigned to the selected slot of the object being inspected.
Edit > Cancel	Cancels any text entry done by the user in the text pane, and return the inspector text pane to its unmodified form (which may be to show the printString of the selected slot).
Session > Commit	Commits the Launcher's session.
Session > Abort	Aborts the Launcher's session.
Session > Begin	Begins a new transaction.
Variables > Inspect	Opens a new inspector on the selected item from the inspector list pane.
Variables > More	Retrieves more of the object's indexable fields. By default, the first 100 are retrieved. Each pick of this menu item doubles the number of fields that are retrieved.

Inspecting Dictionaries and Sets

When you inspect a Dictionary and select an association in the left pane, its value is shown in the right pane. For example, opening the inspector on the `HelpRequestSystem` SymbolDictionary and selecting `#HR_Customer` shows the value, the HR_Customer class.

Figure 4.3 Inspecting a Dictionary

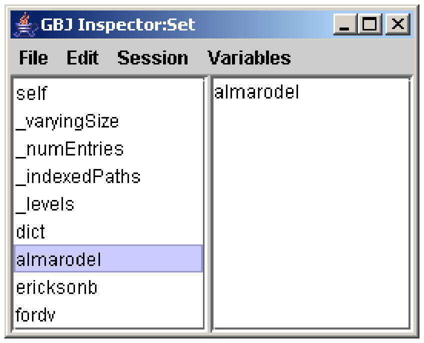


self
numElements
numCollisions
collisionLimit
tableSize
#HelpRequestSystem
#HR_Action
#HR_Company
#HR_Customer
#HR_Engineer

HR_Customer

When you inspect the Set returned by evaluating `HR_Engineer allEngineers`, the elements (the engineers' login names) are shown in the left pane.

Figure 4.4 Inspecting a Set



self
_varyingSize
_numEntries
_indexedPaths
_levels
dict
almarodel
ericksonb
fordv

almarodel

The Debugger

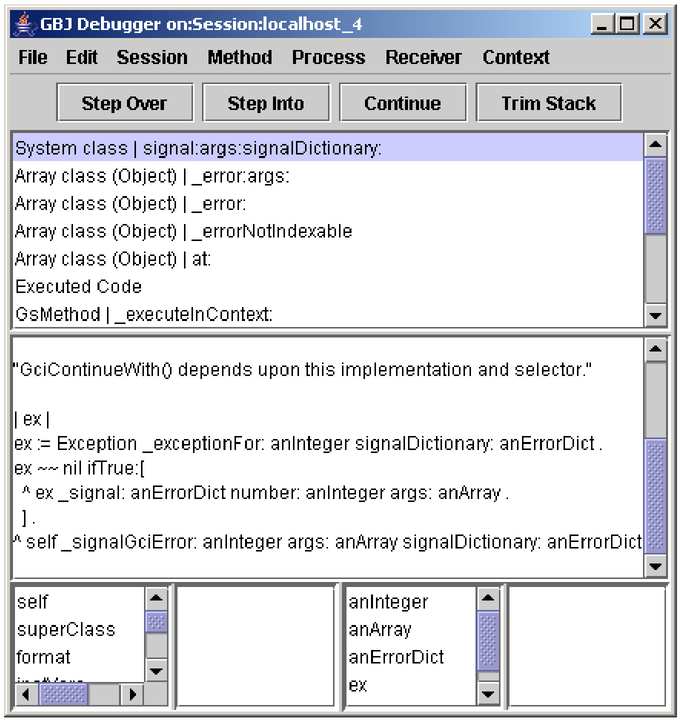
The Debugger can be invoked either from an error notifier or by encountering a breakpoint while running code from a workspace.

For example, the following sequence results from evaluating the expression

```
Array new at: 1
```

with **Do-It** in a workspace. When the error notifier appears, clicking the **Debug** button brings up a Debugger.

Figure 4.5 Debugger


GBJ Debugger on:Session:localhost_4
File Edit Session Method Process Receiver Context
Step Over
Step Into
Continue
Trim Stack

```
System class | signal:args:signalDictionary:
Array class (Object) | _error:args:
Array class (Object) | _error:
Array class (Object) | _errorNotIndexable
Array class (Object) | at:
Executed Code
GsMethod | _executeInContext:
```

```
"GciContinueWith() depends upon this implementation and selector."

| ex |
ex := Exception _exceptionFor: anInteger signalDictionary: anErrorDict .
ex ~~ nil ifTrue:[
  ^ ex _signal: anErrorDict number: anInteger args: anArray .
  ] .
^ self _signalGciError: anInteger args: anArray signalDictionary: anErrorDict
```

```
self
superClass
format
```

```
anInteger
anArray
anErrorDict
ex
```

The debugger consists of six panes. At the top is the *stack pane*, which shows the stack as it was when execution halted do to an error or breakpoint. Select a specific message in the stack, and the *source pane* below shows the source code associated with it.

The bottom four panes are actually two inspectors. The inspector on the left lists the receiver and instance variables of the selected message. Selecting one causes its value to appear in the pane directly to the right. The inspector on the right lists the variables referred to in the selected message. Selecting one causes its value to appear in the rightmost pane.

Debugger Buttons

Four debugger buttons provide convenient access to common debugging functions.

Step Over steps process execution over the next message-send highlighted in the source pane. Control then returns to the debugger with the next step point highlighted.

Step Into steps process execution into the next message-send highlighted in the stack pane. Control typically returns to the debugger with a new message on top of the stack. Execution is halted at the beginning of the new method in the source pane.

Continue continues executing the process and closes the debugger.

Trim Stack trims messages from the stack, starting at the top and trimming until the selected message is reached. The next step point is at the beginning of the new top of the stack.

Debugger Menus

The debugger File, Edit, Session, and Method menus are all similar to those in other tools. The Process menu contains these items:

Step Over	Steps process execution over the next message-send highlighted in the source pane. Control then returns to the debugger with the next step point highlighted. This is the same as clicking the **Step Over** button.
Step Into	Steps process execution into the next message-send highlighted in the stack pane. Control typically returns to the debugger with a new message on top of the stack. Execution is halted at the beginning of the new method in the source pane. This is the same as clicking the **Step Into** button.
Explain	Prints the original descriptive error text in the source pane.
Continue	Continues executing the process and closes the debugger. This is the same as clicking the **Continue** button.

Trim Stack	Trims messages from the stack, starting at the top and trimming until the selected message is reached. The next step point is at the beginning of the new top of the stack. This is the same as clicking the **Trim Stack** button.
Copy Stack	Creates a text representation of the stack, as seen in the stack pane, and places it in the clipboard.

The Receiver and Context menus have one item each:

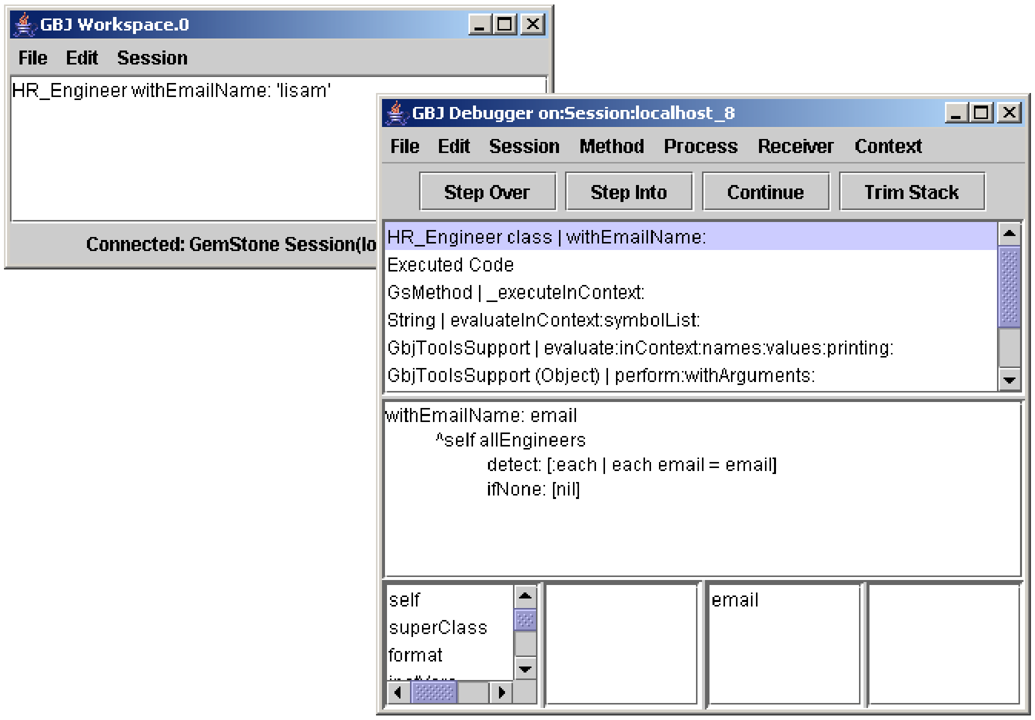
Inspect	Opens an Inspector on the value selected in the leftmost bottom pane, in the case of the Receiver menu, or the bottom pane second from the right, in the case of the Context menu.

Invoking the Debugger from a Workspace

The ordinary way to invoke the debugger is to set a breakpoint in your Smalltalk code, then invoke that code by evaluating an expression in the workspace. For example:

1. Set the breakpoint:
    a. Open any source code browser on the appropriate class, such as `HR_Engineer`.
    b. Select the desired protocol and method.
    c. Place the insertion point before the place where you want execution to halt, then choose **Edit > Set Break**. (To view existing breakpoints, see ”The Breakpoint Browser” on page 3-13.)
2. Open a workspace and enter an expression that invokes code where the breakpoint is set. Select the expression and choose **Edit > Do-It**.
3. When the breakpoint notifier appears, click **Debug**. (**Continue** continues the current process without opening the debugger. **Terminate** ends the current process.

Figure 4.6 Debugging Engineer | withEmailName:



HR_Engineer withEmailName: 'lisam'

Connected: GemStone Session(lo

HR_Engineer class | withEmailName:
Executed Code
GsMethod | _executeInContext:
String | evaluateInContext:symbolList:
GbjToolsSupport | evaluate:inContext:names:values:printing:
GbjToolsSupport (Object) | perform:withArguments:

```
withEmailName: email
	^self allEngineers
		detect: [:each | each email = email]
		ifNone: [nil]
```

Invoking the Debugger from Your Java Client

To invoke the debugger from your running application, you must set a breakpoint within Smalltalk code that is called within a Java `try` block. When the GemStone server encounters the breakpoint, it returns an exception to the client. The catch block uses the exception as an argument to open an instance of `GbjDebugger`. For example, this sequence stops at a breakpoint in `HR_Engineer | email:firstName:lastName:phone:`

1. Find the Java code that calls the Smalltalk code in which the breakpoint will be set. Open a `GbjDebugger` in the `catch` block for that code.

```
try {
   eng = (HR_Engineer) engClass.perform(
      "email:firstName:lastName:phone:", args, 4);
} catch (GbjEventException e) {
   new com.gemstone.tools.GbjDebugger(e);
}
```

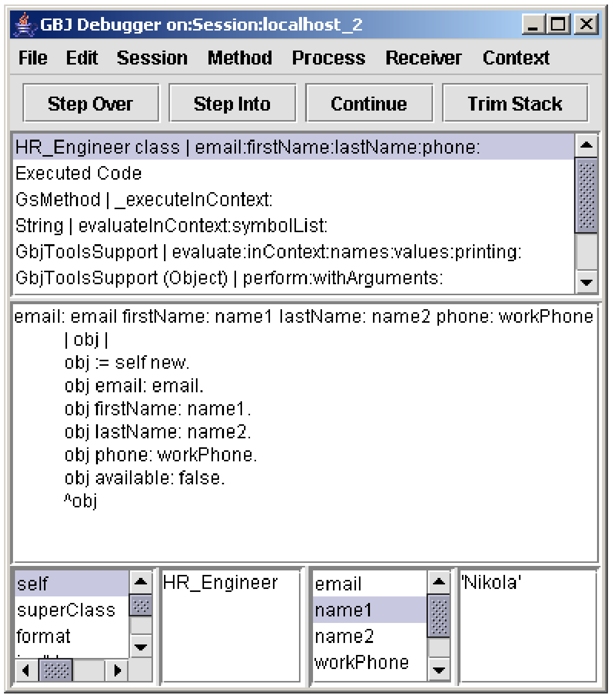
2. Have the application open a `GbjLauncher` so you can set the breakpoint, which must be set within the session in which the application is running. This example opens the Launcher only when the system property `mydebug` is true. The object `session` is an instance of `GbjSession`.

   ```
   if (Boolean.getBoolean("mydebug"))
      new GbjLauncher(session);
   ```

3. Start the application. When the Launcher appears, open a source code browser and set the breakpoint as explained in the previous topic. For our example, the command line is

4. Run the part of the application that calls the Smalltalk code where the breakpoint is set. When the breakpoint is encountered, the debugger will open.

Figure 4.7 Debugger Invoked from a Running Application


GBJ Debugger on:Session:localhost_2
File Edit Session Method Process Receiver Context
Step Over
Step Into
Continue
Trim Stack

```
HR_Engineer class | email:firstName:lastName:phone:
Executed Code
GsMethod | _executeInContext:
String | evaluateInContext:symbolList:
GbjToolsSupport | evaluate:inContext:names:values:printing:
GbjToolsSupport (Object) | perform:withArguments:
```

```
email: email firstName: name1 lastName: name2 phone: workPhone
        | obj |
        obj := self new.
        obj email: email.
        obj firstName: name1.
        obj lastName: name2.
        obj phone: workPhone.
        obj available: false.
        ^obj
```

```
self          HR_Engineer    email        'Nikola'
superClass                   name1
format                       name2
                             workPhone
```

You can step through the Smalltalk execution as desired. Because the client Java thread is not suspended, however, it continues execution and may encounter a

`NullPointerException` on a object that was not returned from the GemStone server.

Common Menus and Shortcuts

All text panes in all GemBuilder for Java tools have a popup menu allowing you to perform common operations. The popup menu differs slightly, depending on which tool the text pane belongs to.

To access the popup menu, place the cursor in a text pane and click the right mouse button. To dismiss the popup menu without selecting an item, move the cursor off the menu and click the left mouse button.

The Popup Menu

The popup menu items are:

Cut	Cuts the selected text and places it in the clipboard.
Copy	Copies the selected text and places it in the clipboard.
Paste	Pastes the text from the clipboard at the current insertion point.
Select All	Selects all the text in the pane.
Do-It	Uses the GemStone compiler to evaluate the currently selected text.
Print-It	Performs a **Do-It**, sends the resulting object the message printString, and then places the resulting string after the current text pane selection.
Inspect-It	Performs a **Do-It**, then opens a GemStone Inspector on the resulting object.
Compile	Compiles the selected source code. This menu item is available only in those tools for which it is relevant.
Cancel	Cancels any text you entered in the text pane, returning it to its unmodified form. This menu item is available only in those tools for which it is relevant.
Set Break	Sets a breakpoint at the current insertion point in the current method. This menu item is available only in those tools for which it is relevant.

Keyboard Shortcuts

The following keyboard shortcuts are available:

Edit > Cut	Control-X
Edit > Copy	Control-C
Edit > Paste	Control-V
Edit > Select All	Control-A
Edit > Do-It	Control-D
Edit > Print-It	Control-P
Edit > Inspect-It	Control-I
Edit > Compile	Control-S
Session > Commit	Control-Shift-S

Chapter 5

Working with GemStone Classes and Methods

About GemStone Classes

Instance Variables Can Be Constrained

GemStone/S 64 Bit does not use constraints, so this section does not apply when running with a GemStone/S 64 Bit server.

To speed GemStone Smalltalk's indexed associative access for efficient querying, you can constrain the value of an instance variable to contain only specified kinds of objects. (Class variables can also be constrained.)

Constraining a variable means that its value is always either an instance of the specified class, a subclass thereof, or nil.

Constraints can be circular. You can constrain an instance variable to be an instance of its own class. You can also constrain instance variables of two classes to each hold instances of the other.

Constraints, like instance variables, are inherited by subclasses: when you define a subclass, its inherited instance variables by default bear the same constraints as specified in the superclass from which they are inherited.

However, inherited instance variables can be further constrained in a subclass. In this case, the instance variable's new constraint must be a subclass of that specified by the inherited constraint.

To further constrain inherited instance variables, specify the name of the inherited variable and its new constraint in the argument to the constraints: keyword in the class definition template. For example, suppose you have defined a class Employee with instance variables named jobTitle and department that are constrained to be Strings. You can now create a subclass of Employee named FormerEmployee and constrain the inherited variables jobTitle and department to be InvariantStrings. FormerEmployee's new instance variables can be constrained or not, as you require, and its other inherited instance variables retain whatever constraints were set in the superclass that defined them.

Instances Can Be Made Invariant

The definition of the class can specify that all instances of the class are invariant. An invariant object can be modified only during the transaction in which it is created. After the transaction is committed, you can no longer modify its instance variables, nor the size or class of the object. You can specify invariance for all instances of a class by providing the argument true to the instancesInvariant: keyword in the class definition template. Class-level invariance is useful for supporting literals in methods and in other limited situations, but it is generally more cumbersome than object-level invariance.

Any object can be made invariant by sending it the message immediateInvariant. This mechanism protects objects from being modified and can be useful for maintaining the integrity of your repository. After immediateInvariant is sent to an object, you can no longer modify its instance variables, nor the size or class of the object. The effect of the immediateInvariant message is not reversible. The message isInvariant returns true if the receiver is invariant; false otherwise.

Reserved Selectors

A small number of selectors are reserved for sole use by the GemStone kernel classes. Do not create new methods with these selectors. If you do, the Smalltalk compiler will not execute your code.

and: downTo:by:do: downTo:do: ifFalse: ifFalse:ifTrue: ifTrue:	ifTrue:ifFalse: isNil notNil or: timesRepeat: to:by:do:	to:do untilFalse untilTrue whileFalse: whileTrue:

Optimized Selectors

Certain selectors are optimized in Smalltalk kernel classes. Redefining an optimized selector in the class for which it is optimized has no effect; the same primitive method will be called and your redefinition will be ignored. However, you can redefine them in other classes, such as those comprising your application.

Class	Selector
SmallInteger	+
SmallInteger	-
SmallInteger	*
SmallInteger	>=
SmallInteger	=
Any kernel class	==
Any kernel class	~~
Any kernel class	_class
Any kernel class	isKindOf:
Any kernel class	_disableProtectedMethod
Any kernel class	_gsReturnNoResult

Defining a New Class

This topic explains how to define a class in GemStone and gives an example. It also discusses several related topics.

To Define the Class

1. Open a GemStone Browser if one is not already open.
2. Make sure no classes are selected in the class list, and, in the Symbol List pane, select the dictionary in which you wish to refer to the new class. The browser responds by displaying a template:

```
NameOfSuperclass subclass: 'NameOfClass'
  instVarNames: #('instVarName1' 'instVarName2')
  classVars: #('ClassVarName1' 'ClassVarName2')
  classInstVars #('ClassInstVarName1'
    'ClassInstVarName2')
  poolDictionaries: #[]
  inDictionary: aDictionary
  constraints: #[]
  instancesInvariant: false
  isModifiable:false
```

3. Replace *NameOfSuperclass* with the name of your new class's immediate superclass. You cannot create subclasses of certain GemStone kernel classes. A note to that effect is included in the descriptions of those classes in the *GemStone Programming Guide.*
4. Replace *NameOfClass* with the name of the new class. By convention, the first character in each GemStone class name is capitalized.
5. Replace *instVarNameX* with the names of any instance variables, or delete all the text within the parentheses if your new class has no instance variables. A class can define up to 255 instance variables to which you can refer by name in that class's methods.
6. Replace *classVarNamesX* with the names of any class variables, or delete all the text within the parentheses if your new class has no class variables. Replace *classInstVarNames* with the names of any class instance variables, or delete all the text within the parentheses if your new class has no class instance variables.
7. Fill in the brackets after the `poolDictionaries:` keyword with any pool dictionaries that you want the class to access.
8. Examine the name of the dictionary after the `inDictionary:` keyword. Unless you replace the inserted text with the name of another symbol dictionary to which you have access, your new class is defined in that dictionary.

9. Fill in the brackets after the `constraints:` keyword with any constraints you wish to specify for one or more of the instance variables. When logged into a GemStone / S 64 Bit server, you should leave it empty. If your class is a collection subclass and you wish to constrain its elements, simply type the name of the constraint class inside the brackets. For example, to constrain a Bag subclass BagOfEmployees to contain only instances of the class Employee, type:

   ```
   constraints: #[Employee]
   ```

10. However, if you wish to constrain a named instance variable, type the name of the variable and the constraint class as a pair, separated by a comma, each within its own set of square brackets, also separated by commas. Preface the instance variable name with a #. For example, to constrain the instance variables name and address of the class Employee to be a String and an instance of the class Address, respectively, type:

    ```
    constraints: #[ #[ #name, String], #[ #address, Address] ]
    ```

11. After the `instancesInvariant:` keyword, specify whether instances of the class are modifiable. The default value is false. Change this to true if you wish instances to be invariant.
12. After the `isModifiable:` keyword, specify whether the structure of the class can be modified. The default value is false. Change this to true if you wish the class to be invariant.

InstVars

Named instance variables are variables whose name is shared by all instances of a class and all instances of its subclasses. Each instance, however, holds a distinct value for the variable

ClassVars

A class variable is a variable whose name and value are shared by a class, all of its instances, its subclasses, and all of their instances. Both class and instance methods of the class and its subclasses can refer to the variable.

ClassInstVars

A class instance variable is a variable whose name and value are shared by a class and all of its instances.

PoolDicts

Pool dictionaries are special-purpose storage structures that enable any arbitrary group of classes and their instances to share information

Example of a Class Definition

Example 5.1

```
Object subclass: 'Engineer'
  instVarNames: #('firstName' 'lastName' 'email' 'phone' 'homePhone' 'beep'
    'cell' 'available')
  classVars: #( 'AllEngineers')
  classInstVars: #()
  poolDictionaries: #[]
  inDictionary: Published
  constraints: #[]
  instancesInvariant: false
  isModifiable: true
```

Here is an explanation of each line:

`Object subclass: 'Engineer'`

Engineer is a direct subclass of Object.

`instVarNames: #('firstName' 'lastName' 'email' 'phone' 'homePhone' 'beep' 'cell' 'available')`

The Engineer class has eight named instance variables: firstName, lastName, email, phone, homePhone, beep, cell, and available.

`classVars: #( 'AllEngineers')`

This class has one class variable, AllEngineers.

`classInstVars: #()`

This class has no class instance variables.

`poolDictionaries: #[]`

This class need not access any pool dictionaries.

`inDictionary: Published`

Engineer resides in the global Published dictionary.

`constraints: #[]`

No constraints have been placed on the instance variables or the class variable.

`instancesInvariant: false`

When instances of the class are created, their values will be modifiable even after they have been committed to the repository.

`isModifiable: true)`

This class is modifiable; instance variables can still be added, removed, and constrained, and class or class instance variables can be added. As long as the class itself remains modifiable, however, no instances of it can be created.

Private Instance Variables

In addition to the private methods discussed earlier, you also see private instance variables occasionally in the GemStone kernel classes. For example, the GemStone Bag class defines four private instance variables that are used by the object manager and primitives to implement features of UnorderedCollections, such as adding indexing structures that alloy efficient querying. The names of private instance variables are symbols beginning with an underscore (_). Private instance variables cannot be modified or constrained when defining subclasses. An attempt to place a constraint on a private instance variable generates an error.

Subclass Creation Methods

You can choose from a variety of subclass creation messages, depending on the type of class you want to create. For example, if you wish to create a byte subclass, replace the initial keyword `subclass:` with the keyword `byteSubclass:`. If the superclass is not a subclass of String, instances of the new class store and return SmallIntegers in the range 0–255.

Similarly, if you wish to create an indexable subclass, replace the initial keyword `subclass:` with the keyword `indexableSubclass:`. Instances of the new class are represented as pointer objects.

For complete descriptions of the different kinds of classes, see the section describing class storage format in the *GemStone Programming Guide.*

If you wish to set the class history of your new class explicitly, you can include the keyword `newVersionOf:` in the class definition template or any subclass creation message, after `instancesInvariant:`. If the argument to this keyword is a class, this method creates the new class as a new version of that class, and the two classes share a class history. In this way, you can make one class a new version of another even if they do not have the same name.

If the argument to the `newVersionOf:` keyword is nil, the new class is created with a new class history.

If you do not include the `newVersionOf:` keyword, the compiler checks to see if another class having the same name already exists. If it does, the new class is compiled as a new version of the other class and shares its class history. If it does not, the new class is created with a new class history.

For more discussion of class histories, see the section describing class histories in the *GemStone Programming Guide*.

Compiling the Class Definition

After you have edited the class template to your satisfaction, choose **Edit > Compile**. When you do this, the GemStone Smalltalk compiler evaluates the class definition template in the browser's text pane as a subclass creation message. For this reason, make sure you don't have any extraneous text in the text pane when you choose **Compile**.

If an Error Occurs

If the class definition does not compile properly, an error message is inserted in the browser's text pane. Delete the error message, then edit your subclass creation message and try to compile the class definition again.

If the new class object cannot be created due to a run-time error, an error notifier window is displayed. Edit the class creation message and try again.

GemStone supports modification of the schema when you already have instances of invariant classes populating your repository and you discover that you must redefine some of your classes.

Modifying an Existing Class

If you select an existing class, then modify and save the class definition, you are creating a new version of the class and all of its subclasses. The browser attempts to recompile all methods from the previous version into the new version. Methods that fail to recompile are presented in a method list browser, from which you can correct the errors. If the class has subclasses, they are also versioned and their methods recompiled.

Versioning a class does not cause its instances to be migrated to the new class. They are still instances of the old class. You can migrate some or all instances of one version of a class to another version when you need to.

For more information on migrating instances, see the section on class histories in the *GemStone Programming Guide*.

To create a new version of a class, select the class's name in the browser to bring up its definition in the text pane. Edit the definition and select **Edit > Compile**. Whenever you create a class with the same name as a class that already exists in one of your symbol dictionaries, the new class is automatically created as the latest version of the existing class and it automatically shares the same class history. Instances created after the redefinition have the new class's structure and access the new class's methods. Instances that were created earlier have the old class's structure and access the old class's methods, but they can be migrated to the new class.

Let's assume that you have a class named Vendor with instance variables for `id`, `name`, `address`, `city`, `state`, `zipCode`, and `phone`. Five of Vendor's instance variables are constrained to be instances of class String, and one (zipcode) is constrained to be a SmallInteger. Suppose that you decide that the class needs an additional instance variable named fax to represent the Vendor's fax number.

To do this, you can define a new version of the class Vendor to include the new instance variable. Keeping the same name as the old class ensures that it shares the same class history as the previous version.

After you compile the class definition, the new class is named Vendor, and all of the original instance and class methods are copied to the new class. Any existing instances will still belong to the original class and may have to be migrated to the new class.

Defining Methods

NOTE:
You can modify only methods for which you have write authorization—for example, methods that you have written for your own classes. You cannot modify any GemStone kernel class method—that is, any method that is defined for one of the predefined classes supplied with the GemStone system.

To define a new method, select the appropriate category and either select no method or execute **Method > New Template**. The following template is displayed for you:

Example 5.2

```
message selector and argument names
    "comment stating purpose of message"

    | temporary variable names |
    statements
```

Replace the template text as indicated. After you have completed the method definition, choose **Edit > Compile**. If the new method compiles successfully, it is inserted into the list of methods for the selected category and class. Otherwise, the first compiler error encountered is inserted into the text at the point of error and becomes the new text selection.

Public and Private Methods

Most of the methods for GemStone kernel classes are public; that is, they are supported methods that appear in the manual. Along with these public methods, you will also see methods whose selector begins with an underscore character (_). These are private methods that are implemented by system developers to support the public protocol. Unlike the public methods, private methods are not supported and are subject to change; they are reserved for use by GemStone developers.

Because Smalltalk is an open system, private methods are listed along with the public methods for the associated category. As with public methods, you can display the text of private methods in the GemStone Browser's text pane. Although it may be interesting to review the definition of private methods, observe the following caution:

CAUTION:
Private methods are subject to change at any time. Do not depend on the presence or specific implementation of these private methods when creating your own classes and methods.

Saving Class and Method Definitions in a File

It's often useful to store the GemStone Smalltalk source code for your classes and methods in ordinary files. This process is called filing out source code. Such files make it easy to:

- transport your code to other GemStone systems,
- perform global edits and recompilations,
- produce paper copies of your work, and
- recover code that would otherwise be lost if you are unable to commit.

As noted earlier, the GemStone Browser provides File Out menu items so you can file out methods and class definitions.

Reading and Compiling a Saved File

- Open a GemStone workspace, or click on the Launcher.
- Select **File > File In**.
- Select the file in the resulting file dialog.
- Click **Open**.

Saved files are written by GemStone as sequences of commands such as those understood by Topaz, the command line-oriented version of the GemStone programming environment.

Sample File-out Contents

This example shows a class definition in Topaz format.

Example 5.3

```
! Class 'Address'
!
run
Object subclass: 'Address'
        instVarNames: #( 'street' 'zip')
        classVars: #()
        poolDictionaries: #()
        inDictionary: UserGlobals
        constraints: #[#[#street, String],
                    #[#zip, Integer]]
isInvariant: false
%
!
! Instance Category 'Updating'
!
category: 'Updating'
method: Address
street: newValue
    "Modify the value of the instance variable5 'street'."
     street:= newValue
%
method: Address
zip: newValue
    "Modify the value of the instance variable 'zip'."
     zip:= newValue
%
!
! Instance Category 'Accessing'
```

```
!
category: 'Accessing'
method: Address
street
    "Return the value of the instance variable 'street'."
     ^street
%
method: Address
zip
    "Return the value of the instance variable 'zip'."
    ^zip
%
```

GemStone's filing out and filing in facilities are intended mainly for saving and restoring classes and methods without manual intervention. If this is all you want to do, then you don't need to understand the Topaz commands involved. However, it is also possible to create custom files that include commands to commit transactions and to create and manipulate objects other than classes and methods. If you want to perform such tasks, you'll need a full description of the Topaz commands. See the *Topaz Programming Environment* manual for your system.

The file-in mechanism, however, has certain limitations. It cannot execute the full set of Topaz commands. Instead, it is restricted to the following subset:

category:	**method:**
classmethod	printit
classmethod:	removeAllMethods
commit	removeAllClassMethods
doit	run
method	

GemStone file-in mechanism acknowledges the presence of the following commands by adding notes to the transcript, but it does not execute them:

display	omit
expectvalue	output
level	remark
limit	status
list	time

If GemStone encounters any other Topaz commands, it stops reading the file and displays an error notifier.

The file in mechanism does not display execution results, either. Instead, it appends information to the transcript about the files it reads and the classes and categories for which it compiles methods.

Index