GemStone®

GemBuilder for Smalltalk Tutorial

March 2005

GemStone/S

For use with GemBuilder for Smalltalk Version 6.1
and Cincom VisualWorks Version 7.3

Preface

About This Tutorial

This tutorial is a task-oriented introduction to writing applications using the GemStone/S object-oriented database and GemBuilder for Smalltalk®. It is organized in four lessons, each of which concentrates on a specific aspect of application development. When you have completed the lessons, you will have worked through a basic Smalltalk application for the GemStone/S object server.

This tutorial introduces you to the basics of the GemBuilder interface and code development tools. Use it with your own private copy of the GemStone/S object repository, so that you can feel free to explore as you wish without impacting the work of others, or your own later work.

This tutorial is intended to work with a VisualWorks image, Version 7.2 or later, in which GemBuilder for Smalltalk Version 6.1 or later has been installed.

Intended Audience

This tutorial assumes that you are familiar with the Smalltalk programming language and user interface, and that you have a basic understanding of the principal database concepts. It further assumes that your system meets the requirements listed in the installation section of the *GemStone/S Installation Guide*, that GemStone/S system has been correctly installed on your host computer as described in that guide, and that GemBuilder for Smalltalk has been installed in your VisualWorks image on your host computer according to the instructions in the *GemBuilder for Smalltalk Installation Guide*.

Organization

An introduction describes the example application and lists the included files. The tutorial contains the following lessons:

- Lesson 1 shows how a GemBuilder for Smalltalk image differs from the unmodified VisualWorks Smalltalk image and points out some of its unique and helpful features.
- Lesson 2 explains how to connect an object in the Smalltalk client image with an object in the database so that changes to one are reflected in the other.
- Lesson 3 shows how GemStone users are represented by user profiles and how more than one user can access the same object using a symbol list dictionary.
- Lesson 4 shows how to query the repository.

Documentation Conventions

We use the following typeface and vocabulary conventions in order to keep the instructions concise and clear:

- Key names are enclosed in **KeyFont**. For example, the Return key is indicated by **Return**.
- Menu items appear in **MenuFont**. For example, the “accept” menu item appears as **accept**. When menu items cascade to submenus, the menu and submenu items both appear, separated by a right-arrow symbol (>). For example, the **save** submenu item of the **special** menu item is written as **special > save**.
- This tutorial refers to mouse buttons according to the default Smalltalk conventions: the left mouse button is the *select* button and the right mouse button is the *operate* button. If you have remapped your mouse buttons, make the appropriate adjustments as you follow the instructions.
- Work in the repository is done using GemStone Smalltalk, the GemStone programming language. Work in GemBuilder for Smalltalk is done using VisualWorks Smalltalk. Text that refers to *the Smalltalk image* refers to a VisualWorks image in which GemBuilder for VisualWorks has been installed.

- Execution in GemStone Smalltalk is accomplished using the menu items **GS-Do it**, **GS-Print it**, or **GS-Inspect it**. Execution in the client Smalltalk is accomplished using the menu items **Do it**, **Print it**, or **Inspect it**. Unless further amplified, the expression "execute in GemStone Smalltalk" is interchangeable with "execute using **GS-Do it**," and the expression "execute in the client Smalltalk" is interchangeable with "execute using **Do it**."
- In Smalltalk examples of either kind, client or GemStone code appears in a `monospace` typeface. The values returned from either kind of Smalltalk execution, such as those returned from a **Print it** or **GS-Print it** expression, appear `underlined`. For example:

```
(PimUserProfile allUserProfiles at: 'PimRoot') scheduleitems ==
(PimUserProfile allUserProfiles at: 'PimUser1') scheduleitems
                  true
```

Other Useful Documents

While developing applications for GemStone/S, you will probably need to consult other GemStone documentation. A list of other GemStone documentation is provided at the end of the last lesson.

If you wish to learn about Smalltalk programming, see *Smalltalk-80: The Language and its Implementation*, by Goldberg and Robson (Addison-Wesley, 1983), and *Smalltalk-80: The Interactive Programming Environment*, by Goldberg (Addison-Wesley, 1984).

For good general discussion of object-oriented design, see *Object-Oriented Design with Applications* by Grady Booch (Benjamin Cummings Publishing, 1991) or *Designing Object-Oriented Software*, by Wirfs-Brock, Wilkerson, and Wiener (Prentice Hall, 1990).

For discussion of object-oriented databases, see *Object-Oriented Databases*, by Chorafas and Steinmann (Prentice Hall, 1993), *Object Data Management*, by Catell (Addison Wesley, 1991), or *Object-Oriented Concepts, Databases, and Applications*, edited by Kim and Lochovsky (ACM Press, 1989).

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 Smalltalk is provided in PDF format. This is the same documentation that is included with your GemBuilder for Smalltalk 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/S 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 for GemStone/S 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

List of Figures

Chapter

Introduction to the Example Application

In this tutorial, you will be working with an example application: a Personal Information Manager that keeps track of your calendar, your to-do list, and a list of contacts. A set of classes with their accompanying methods implements the application in VisualWorks; it is provided in parcel format as the files *GsPim.pcl* and *GsPim.pst* in your *tutorial* distribution directory.

To save you the labor of typing as you work through the exercises (as well as possible confusion resulting from typographical errors), many of the expressions you will need are provided in a file, also in the *tutorial* directory, named *tutorialWorkspace.txt*. If you don't wish to type, open this file in a File Browser, then execute in client or GemStone Smaltalk.

In addition, the tutorial directory contains a small class definition—*GemStoneAdministrator.gs*— and a file containing just its methods—*GemStoneAdministrator-methods.gs*—for practice filing in GemStone code and managing class versions.

With these building blocks you will create and modify the application in GemStone/S.

Each lesson is designed to highlight specific features of the GemStone/S repository. As you work through this tutorial, you will first explore useful features of the GemBuilder for Smalltalk interface. Then you will persist appropriate parts of the example application in GemStone/S and create GemStone/S users who can view and add to the data. Finally, you'll modify the application so that data can be shared and users can query it.

Naturally, within one tutorial application, we can only touch on aspects of these features: for deeper and more thorough discussions, consult the other GemStone documentation described at the end of the last lesson.

In one critical matter, however, the tutorial application is probably unlike an application you are likely to write. This tutorial can be run by one person using a single host machine and a single VisualWorks image. However, more than one imaginary user has to share that image. This is not the ordinary situation—more commonly, each GemStone user runs his or her own Smalltalk image (or other interface). Several users sharing one image has led to some less-than-perfect compromises with respect to connectors. If you take the final lesson to its obvious conclusion, you'll get a connector error. This imperfection is the inevitable result of using a tool (in this case, the Smalltalk image) in a way that it was not intended to be used. We hope you do not find this confusing; we have done our best to explain the cause as well as providing a work-around.

One final caution: this tutorial is not intended to be an actual production application. Your Personal Information Manager users' default passwords are `password`. Needless to say, we do *not* recommend this approach to system security.

Happy coding!

Chapter 1

A Tour of GemBuilder for Smalltalk

This lesson aims to make Smalltalk programmers more comfortable with GemBuilder for Smalltalk (GBS) and able to manage its principal differences from your client Smalltalk. It introduces you to the most important differences between GBS and the client Smalltalk image as it is shipped.

This lesson assumes that you are familiar with the Smalltalk programming language, and that you have already installed GBS according to the instructions provided in your GemBuilder Release Notes. For more information about GBS, see the *GemBuilder* manual for your client Smalltalk.

1.1 Objectives

When you have finished this lesson, you will be able to:

- log into GemStone from GBS, start a transaction, end a transaction, and log out from GemStone;
- understand the messages that GemStone prints to the Transcript, and know how to turn them on or off;
- understand the difference between executing GemStone code and client Smalltalk code, and when it is appropriate or required to do each;

- use the special inspectors GemStone provides to inspect nonsequenceable collections;
- create and file in GemStone code, and manage class versions; and
- understand the mixed GemStone and client Smalltalk contexts in the debugger, and set or clear GemStone method breakpoints.

1.2 Logging In and Out, and Managing Transactions

This exercise teaches you how to use the GemStone Session Browser to log into or out of GemStone, and to commit or abort transactions.

Logging Into GemStone

Step 1. Start your image as specified in the *GBS Installation Guide*.

Step 2. Before you can log in to a GemStone repository, you need to specify the name and path to the appropriate shared library for that version of GemStone. Provided the shared libraries are on your operating system search path, you can set this by opening a VisualWorks workspace and executing the following code:

On Windows:

```
GbsConfiguration current libraryName: 'gcilw61.dll'
```

On Linus and Solaris:

```
GbsConfiguration current libraryName: 'libgcilnk61.so'
```

On HPUX:

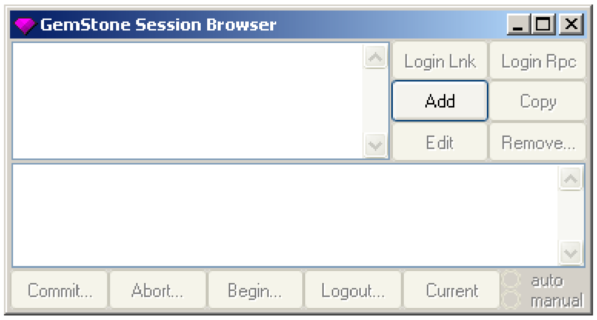
```
GbsConfiguration current libraryName: 'libgcilnk61.sl'
```

After doing this, save your image.

Under some installations, you may need to use a different library name, or include the path; in this case please contact your GemStone system administrator.

Step 3. From the GemStone menu, execute the menu item **Tools > Browse Sessions** or select the toolbar icon. The GemStone Session Browser appears.

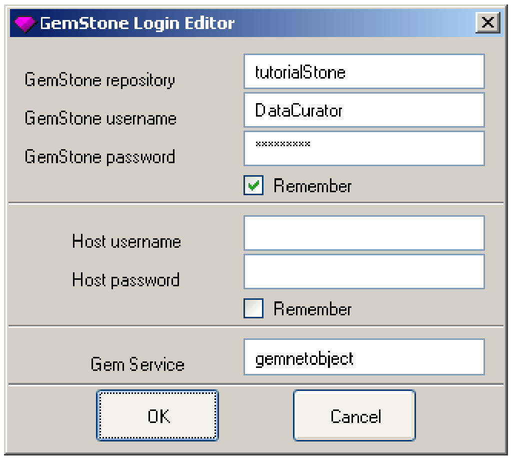
FIGURE 1.1 The GemStone Session Browser



Step 4. Click the button labeled **Add**.

Step 5. A new window appears, prompting you for the session parameters. Enter your own repository's Stone name. Instead of providing your own GemStone username, type **`DataCurator`**. Instead of providing your own password, type **`swordfish`**. Click **Remember**, then **OK**. The window disappears, and the session parameters you have provided appear in the top left pane.

FIGURE 1.2 Providing Session Parameters


GemStone Login Editor

GemStone repository	tutorialStone
GemStone username	DataCurator
GemStone password	*********
	[✓] Remember
Host username	
Host password	
	[ ] Remember
Gem Service	gemnetobject

Step 6. Back in the Session Browser, select the session parameters you provided in the previous step. The line highlights when you click on it, and the buttons to the right are enabled.

Step 7. Click on the button labeled **Login Lnk**, if it is available, or **Login Rpc** if it is not. When the session appears in the session (middle) pane, you are logged into GemStone. Most buttons at the bottom of the Session Browser are now also enabled, and the following message (or a similar one for a remote session) prints in the System Transcript:

```
Logged in Session 1 (linked) for 'DataCurator' on 'yourStoneName'
```

Leave the Session Browser open and leave yourself logged in for the next exercise.

Committing a Transaction

Now we will create a new symbol dictionary, an object you can use to store an application schema and data. Symbol dictionaries are a mechanism for GemStone to allow more than one user to access an object at the same time. We will use another for the tutorial application. Having a whole application stored in its own dictionary makes it easier to reclaim all the storage used by the tutorial objects after they are no longer of interest to you.

Step 1. From the GemStone menu, open a Classes Browser by executing **Browse > All Classes** or select the toolbar icon.

Step 2. The top leftmost pane of the Classes Browser lists symbol dictionaries. Choose **add** from the operate button popup menu.

Step 3. In the resulting dialog, enter **`UserClasses`**.

Step 4. Open a workspace, using the toolbar icon. Move and frame the workspace as you require.

Step 5. In the workspace, type the following:

```
System myUserProfile insertDictionary: UserClasses at: 1.
```

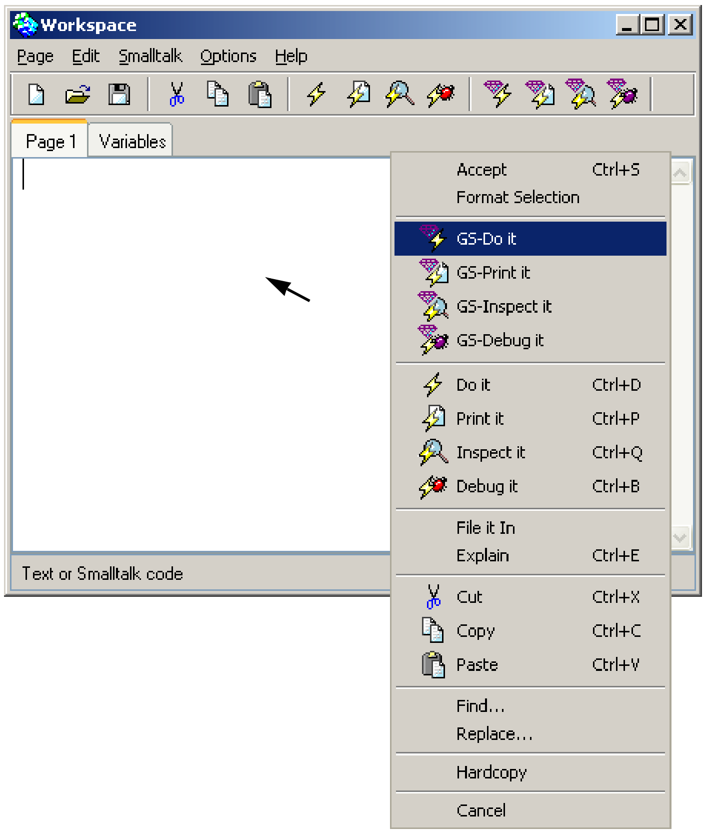
This line adds the new symbol dictionary to your user profile. (We will explore symbol dictionaries and user profiles in more detail later.)

Step 6. Select the text that you just typed in the workspace, and bring up the operate button menu.

Notice that the menu has several new items: **GS-Do it**, **GS-Print it**, **GS-Inspect it**, and **GS-Debug it**. We will explore these items in greater detail in the next exercise.

Execute the menu item **GS-Do it**.

FIGURE 1.3 The Workspace Menu



Step 7. In the Session Browser, select the current session listed in the bottom list pane, if necessary, to enable the bottom row of buttons.

Step 8. Click the button on the bottom row labeled **Commit...**, and respond to the confirmer by clicking on **yes**. Notice that a message prints in the Transcript, confirming that the transaction has been committed.

```
Session 1 (linked) for 'DataCurator' on 'yourStoneName' committed
transaction at 2:04:15 pm.
```

Your new dictionary is now part of your GemStone database, available for sharing.

Aborting a Transaction

Step 1. In the same workspace, type:

```
System myUserProfile removeDictionaryAt: 1.
```

Step 2. Select the text you just typed and execute the menu item **GS-Do it**. This removes the new dictionary from your symbol list.

Step 3. In the Session Browser, click the button on the bottom row labeled **Abort...**, and respond to the confirmer by clicking on **yes**. After all, we don't really want to remove this dictionary yet.

Your new dictionary remains part of your GemStone database.

Logging Out of GemStone

Step 1. In the Session Browser, select the current session listed in the bottom list pane if it is not already selected.

Step 2. Click the bottom button labeled **Logout...**.

Step 3. Respond to the confirmer by clicking on **No**. Because you have done no useful work since the last transaction you committed, you have no need to commit this transaction.

The transcript faithfully echoes this latest change as well:

```
Logging out Session 1(linked) for 'DataCurator' on 'yourStoneName'
```

Managing the Transcript

In addition to logging in and out, committing transactions, or aborting them, the Transcript can echo changes in transaction mode, code filing in or out, and classes being generated. You can turn off these messages by executing the client Smalltalk code:

```
GBSM verbose: false
```

You can turn them on again by executing the client Smalltalk code:

```
GBSM verbose: true.
```

For the tutorial, however, leave `GBSM verbose` set to `true`.

Log back into the database when you are ready to continue with the tutorial. Now that you know the basics of logging in and out and managing transactions, we can touch on some deeper differences between GemStone and client Smalltalk.

1.3 GemStone and Smalltalk Execution

Although a Smalltalk image in which you have installed GBS looks very much like an ordinary Smalltalk image, it has one key feature that fundamentally changes the way you work. We have already touched on this feature in the previous exercise; indeed, it is difficult to avoid it. A client Smalltalk image that includes GemBuilder for Smalltalk is managing two different worlds: client Smalltalk and GemStone Smalltalk. It incorporates two different name spaces, two different execution engines (for client Smalltalk and GemStone Smalltalk), and two different sets of tools. Indeed, you are working with two different images: the client Smalltalk image in your computer's memory, and a multiuser persistent server image—GemStone—on disk.

Sometimes these two worlds are blended so you can look at everything at once, and sometimes, as with the Smalltalk and GemStone System Browsers, they are presented as entirely separate. And sometimes it is up to you to determine which world you need to enter and which tools are appropriate to accomplish your aims.

For example, you are probably already familiar with Smalltalk workspaces. GBS adds new menu items to this menu, so you can use it for executing client Smalltalk code as you always have, you can also execute GemStone Smalltalk code, if you are logged into GemStone. Log back into GemStone and open one now, if a workspace is not already open.

Step 1. If necessary, open the Session Browser and log in again as DataCurator.

Step 2. After you are logged in, open a workspace, if necessary.

Step 3. In the workspace, type **`Array new: 4.`**

Step 4. Invoke the operate button menu. As you saw earlier, it includes a number of additional items: **GS-Do it**, **GS-Print it**, **GS-Inspect it**, **GS-Debug it**, and **GS-File it In**. They are directly comparable to the ordinary Smalltalk workspace menu items **Do it**, **Print it**, **Inspect it, Debug it,** and **File it In**. The menu item **GS-Do it** invokes the GemStone Smalltalk compiler instead of the client Smalltalk compiler; the menu item **GS-Print it** invokes the GemStone Smalltalk compiler and prints the result; and so on. These menu items are only enabled when you are logged in to the GemStone server.

Step 5. Select the text you typed. `Array new: 4` works in both client Smalltalk and GemStone Smalltalk. First execute **print it**. The client Smalltalk execution cursor appears—an arrow with a star—and the compiler prints `#(nil nil nil nil)` in the workspace.

Step 6. Now select it again and execute **GS-Print it**. The GemStone Smalltalk execution cursor appears—a gem—and the compiler prints `anArray( nil, nil, nil, nil)` in the workspace (its `printString` method works differently).

The two different execution cursors are not surprising in this context, but on occasion the cursor can be a detail worth noticing, because under less obvious circumstances it can provide insight into what is really happening.

Step 7. In the workspace again, type **`Smalltalk`**.

Step 8. Select it and execute **print it**. The word Smalltalk is echoed—delete it.

Step 9. Select `Smalltalk` again and execute **GS-Print it**. The words `undefined symbol->Smalltalk` appear. The GemStone Smalltalk compiler doesn't know anything about client Smalltalk global variables, including the global Smalltalk.

Step 10. In the workspace again, type **`UserClasses`**.

Step 11. Select it and execute **GS-Print it**, and notice the gem cursor again. The string `aSymbolDictionary(aSymbolDictionary)` appears. We met symbol dictionaries briefly in the previous exercise, when we made our new UserClasses dictionary. They are a mechanism for GemStone to allow more than one user to access an object at the same time.

A symbol dictionary contains any number of keys—names—which it associates with values—specific objects. The dictionary, and therefore the objects it refers to, can be made accessible to all users of the system, to you alone, or to a specific subset of users.

Each GemStone user has a symbol list: an array of symbol dictionaries. By default, each user has access to three: Globals, UserGlobals, and Published. The dictionary Globals refers to all the GemStone kernel classes; it is accessible to all users of the database, so that they can each use the same class Array, for example. The dictionary UserGlobals is unique for each user. You can use it to refer to objects that you create and use, but which no one else needs to access. The dictionary Published is for objects you or others create that are intended to be globally accessible.

If you are developing an application with a group, you can make a special symbol dictionary for the application and add it to the symbol list of each user in the group, as we will do in this tutorial.

For example, we will make another symbol dictionary to refer to all the objects in the tutorial example application. We'll do this to make it easier to reclaim storage later, but you can also add this symbol dictionary to another user's symbol list if you wish to share the tutorial with someone else.

Step 12. Select UserClasses again and this time execute **Print it**. Unsurprisingly, the client Smalltalk does not recognize this new, undefined symbol, and therefore prompts you to declare it in some manner. Click on **Cancel** from the dialog that pops up.

Step 13. It's also possible for the same code to produce two different outcomes, due to the two execution engines. For example, in the workspace again, type **`1 to: 220`**.

Step 14. Select it and execute the operate button menu item **Inspect it**. As the title bar of the Inspector window reveals, the client Smalltalk produces an instance of class Interval. Explore the inspector as you wish and close it when you're done.

Step 15. Select `1 to: 220` again and execute **GS-Inspect it**. GemStone Smalltalk produces an instance of class Array. Explore the inspector as you wish and close it when you're done.

Although your workspace looks like one seamless window, it has a dual nature, like the image of which it is a part. Within it, you can execute client Smalltalk code for your application and GemStone Smalltalk code to access the repository.

1.4 Inspecting GemStone Objects

As you've seen, the workspace menu item **GS-Inspect it** invokes GemStone inspectors instead of client Smalltalk ones. In most cases, these behave similarly to the Smalltalk inspectors already familiar to you (although they will usually be inspecting GemStone objects). This is not true, however, for inspectors on nonsequenceable collections. To see the difference, let's inspect an instance of Bag and an instance of Set in both client Smalltalk and GemStone Smalltalk.

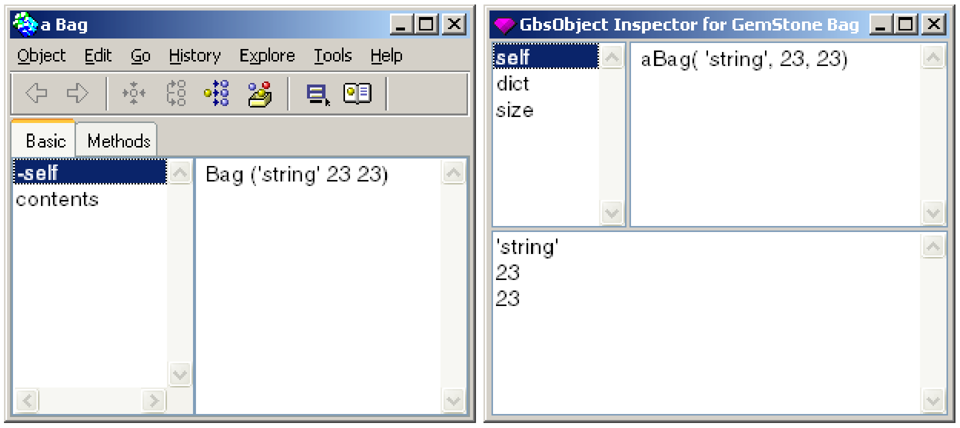
Step 1. In a workspace, type:

```
Bag with: 'string' with: 23 with: 23.
```

Step 2. Select the text you just typed and execute the operate button menu item **Inspect**. This invokes a client Smalltalk inspector on the instance of Bag you created. It contains two items in the left pane. The first, `self`, is the bag itself. The second, `contents`, reveals that Bags are implemented as dictionaries.

Step 3. Select the text again, and this time execute **GS-Inspect it**. (This code also works in either client Smalltalk or GemStone Smalltalk.) The resulting inspector has three panes instead of two. The top left pane contains three items: `self` (the Bag instance), `dict` (the dictionary that implements it), and `size` (the bag contains three items). The bottom pane contains the objects you placed in the bag.

FIGURE 1.4 Smalltalk and GemStone Bag Inspectors


a Bag
Object Edit Go History Explore Tools Help
Basic Methods
-self
contents
GbsObject Inspector for GemStone Bag
self
dict
size

```
Bag ('string' 23 23)
```

```
aBag( 'string', 23, 23)
```

```
'string'
23
23
```

Step 4. In the bottom pane, select the string and invoke the operate button menu item **inspect** to open another GemStone inspector on the string. Its constituent characters are now available for further manipulation, if you wish.

Notice that, in addition to **inspect**, when an item is selected in the bottom pane, the bottom pane menu also allows you to add objects to the bag, remove objects from it, or update the inspector as a whole in case you have made changes to the bag from another window.

Step 5. Close all the inspectors.

Step 6. In a workspace, type:

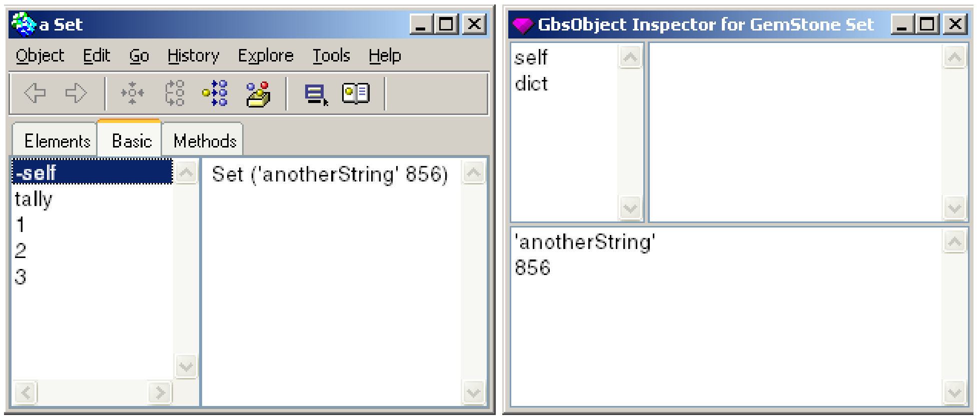
```
Set with: 'anotherString' with: 856.
```

Step 7. Select the text you just typed and execute the operate button menu item **Inspect** to invoke a client Smalltalk inspector on the instance of Set you created. The Set inspector has an additional tab, “Elements”, along the top. Click on the tab “Basic” to see the underlying structure of the Set. This tab contains a two-pane inspector containing five items: `self` (the Set instance), `tally` (the number of items in the set, which is `2`), and *three* objects that it contains, appearing as `1`, `2`, etc. The object `1` has the value `nil`, `,2` has the value `anotherString`, and `3` holds the integer you put into the set.

Step 8. Select the text again, and this time execute **GS-Inspect it**. (This code also works in either client or GemStone Smalltalk.) The resulting inspector has three panes instead of two. The top left pane contains two items, `self` and `dict`. The bottom pane contains the objects you placed in the set. When an item is selected, the same operate button menu appears here as you saw in the Bag inspector's bottom pane.

Step 9. Close both inspectors.

FIGURE 1.5 Smalltalk and GemStone Set Inspectors


a Set
Object Edit Go History Explore Tools Help
Elements Basic Methods
-self
tally
1
2
3
Set ('anotherString' 856)
GbsObject Inspector for GemStone Set
self
dict
'anotherString'
856

Step 10. In a workspace, type:

```
Set new addAll: (1 to: 220); yourself.
```

Step 11. Open a client Smalltalk inspector on this new set. The client Smalltalk inspector **Elements** tabs shows the contents ordered alphabetically, and grouped into ranges. The **Basic** tab also shows ranges, but in numerical order. Also, item 1 received the value nil, so that all items have values that are off by one. Items 222 through 331 are all nil.

Step 12. Open a GemStone server inspector on this new set. The GemStone inspector shows items 1 through 100 in the bottom pane. If you inspect each, you will find it has the value you expect. More to the point, however, the operate button menu in the bottom pane has an additional menu item: **more**.

Execute it, and it doubles the number of items visible in the lower pane, to 200. Execute it again; it would again double the number of items shown, but the set has only 20 more. They become visible (scroll the pane down to see them), and **more** is no longer on the menu, because the entire set is now visible.

Step 13. Close the inspectors when you have finished exploring them.

GemStone provides different inspectors for nonsequenceable collections because collections of objects are a natural way to represent data in a database, and are therefore central to GemStone applications. GemStone provides tools that are optimized for tasks commonly associated with database applications. Experiment further with these inspectors if you wish, and when you are finished, feel free to close both inspectors and the workspace; this lesson makes no further use of them.

1.5 Creating and Filing Code In and Out

In this exercise, we'll create a test class and experiment with some of the conveniences provided by the GemStone Classes Browser.

Step 1. Open a GemStone Classes Browser. From the GemStone menu, execute **Browse > All Classes**.

Step 2. Unlike the VisualWorks System Browser (the Refactoring Browser), which lists categories, packages, or parcels, the top left pane of the GemStone Classes Browser lists symbol dictionaries. You'll see at least Globals, UserGlobals, Published, and UserClasses—the symbol dictionary you made earlier.

Another difference between the GemStone and client Smalltalk System Browsers is that the GemStone Classes Browser includes **commit** and **abort** menu items in the dictionary pane operate popup menu.

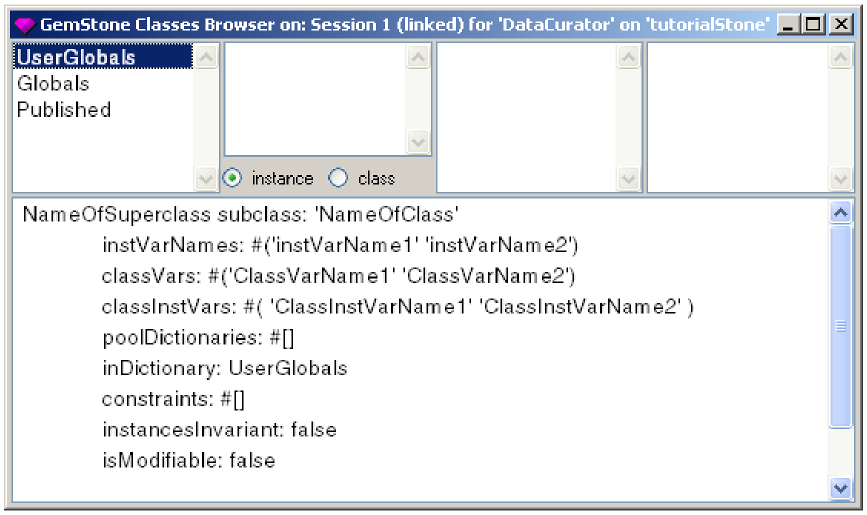
Step 3. Select UserClasses. The class creation template appears in the text pane below. It probably looks familiar to you; it's quite similar to the class creation template available in the Smalltalk System Browser. However, as you've probably guessed, classes created using the GemStone Classes Browser are

GemStone Smalltalk classes, not client Smalltalk classes. Therefore, a few items are different:

- The name of the class being created is delimited by single quotation marks instead of being preceded by a # sign.
- A keyword is present to specify contraints on the values of the instance variables.
- Another line specifies the symbol dictionary that will refer to the class. This line, beginning with the keyword `inDictionary:`, is already filled in with the name of the selected dictionary—in this case, UserClasses.

Naturally, this is editable, but for this exercise we will accept the default, placing the new class in UserClasses.

FIGURE 1.6 GemStone Class Creation Template


GemStone Classes Browser on: Session 1 (linked) for 'DataCurator' on 'tutorialStone'
UserGlobals
Globals
Published
instance
class

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

For practice in creating GemStone Smalltalk classes, as well as to highlight a few conveniences provided by the GemStone Classes Browser, we'll create a small utility class called GemStoneAdministrator. This class allows you to automate repository backups by generating a file name guaranteed to be unique.

Step 4. With UserClasses still selected, select `NameOfSuperClass` and replace it with `Object`.

Step 5. Select `NameOfClass` and replace it with `GemStoneAdministrator`.

Step 6. Double-click inside the parentheses after `instVarNames` and delete them. This class has no instance variables.

Step 7. Repeat the previous step with `classVars` and `classInstVars` to delete those variables as well.

Step 8. Still in the text pane, execute the operate menu item **accept**. The new class has now been created.

Step 9. From the operate button popup menu in the dictionary pane, commit the transaction.

Step 10. Like the client Smalltalk System Browser, the GemStone Classes Browser lets you file out code. Return to the Classes Browser and select the class GemStoneAdministrator, if it isn't still selected.

Step 11. Still in the class pane, access the operate menu. Notice that you can file out an entire class definition with the item **file out as...**, or just its methods with the menu item **file out methods as...**. Execute the item **file out as...**. The resulting dialog prompts you for a file name; the default is the class name with `.gs` appended. Cancel the operation.

Step 12. The GemStoneAdministrator class has various methods. In the message category pane, execute the operate popup menu item **add**. A dialog appears to receive the new category name. Enter `utilities`.

Step 13. In the method template pane, select the template text and type the following method, which uses today's date to create a string that will be part of the backup file name, thus guaranteeing its uniqueness:

```
todaysDateString
        ^Date today asStringUsingFormat: #(2 1 3 $_ 1 2)
```

When you are finished, execute **accept**.

Step 14. The GemStoneAdministrator class has several more methods, and you may not wish to type in all of them. To file in the class and its methods, open a File Browser.

Step 15. Point the File Browser to the *tutorial* subdirectory of your GemBuilder installation directory, and list the files it contains.

Step 16. The files in the tutorial directory appear in the top right pane. Select the file *GemStoneAdministrator.gs*.

Step 17. In the top right pane, hold down the operate menu button. Notice that since the extension *.gs* is not a Smalltalk code extension, there is no menu item **File In...**, but the menu does have an extra item: **GS-File In**.

Step 18. With the file *GemStoneAdministrator.gs* selected, execute **GS-File in**. The gem cursor appears and the class files in; you can see a message to that effect in the System Transcript. Leave the File List open—we'll need it again soon.

Step 19. Return to the Classes Browser (refresh it if necessary), and you'll see that you now have two versions of the class GemStoneAdministrator: the [2] appended to the class name indicates that the one you just filed in is the second version. Because the two versions are identical, this duplication is pointless.

Step 20. Return to the Session Browser and abort the current transaction to rid yourself of the needless duplicate.

Step 21. You now have the class definition but none of the methods. Return to the File List and this time, select the file *GemStoneAdministrator-methods.gs*. This file contains only the methods for the class, not the class definition itself. Execute **GS-File in** to file in the methods.

Step 22. In the Classes Browser, select the class (update the browser if necessary), and take a moment to explore the methods.

Step 23. Select the method `setPathFrom:` in the category `private` and change the UNIX-based default `~/` to a more reasonable path for your system—where you want the backup file to be written.

Step 24. Do the same for the methods `backupDevelopmentServer` and `backupProductionServer` in the category `backup`.

Step 25. Commit your transaction.

Step 26. To use the new class, in a workspace, execute using **GS-Do it**:

```
UserClasses at: #Administrator put: GemStoneAdministrator new
```

This makes a global symbol named Administrator and puts it in your new symbol dictionary. Since it is a symbol, rather than a class, it does not show up in the list of classes in this SymbolDictionary. To see it, use the operation menu item inspect to inspect the SymbolDictionary.

Step 27. Commit the transaction.

NOTE
The following step assumes that you are working on your own private copy of the repository, and that it is small. If that is not true, the next step will take a lot of time and disk space, and you may prefer to skip it.

Step 28. Again in the workspace, execute using **GS-Do it**:

```
Administrator backupDevelopmentServer
```

This backs up your development repository. You will probably want to delete the file after you've logged out.

As you can see, creating classes and methods in GemStone Smalltalk is very similar to the process you use to do in client Smalltalk. The chief difference is that GemStone, being a multiuser system, requires you to choose the symbol dictionary that will hold the reference to the class. The symbol dictionary is the mechanism by which you can control an object's visibility to other users, allowing you to share code and data when needed, and to prevent mix-ups when sharing isn't needed.

In addition to the critical choice of symbol dictionary. the GemStone Classes Browser provides conveniences for automatically compiling accessing and updating methods for variables, and for creating comparable classes in client Smalltalk and mapping them to GemStone Smalltalk classes. We will explore these conveniences in later exercises. For now, explore the Classes Browser as you wish, and close it when you're ready to begin the next exercise.

1.6 Debugging in GemStone

You have now seen many of the special facilities in GemBuilder for Smalltalk. Only two more remain unexplored: the ability, with the debugger, to view GemStone Smalltalk and client Smalltalk contexts mixed together in the stack, and the ability to set and clear two kinds of breakpoints without modifying source code. In this exercise, we explore these features of debugging in GemStone.

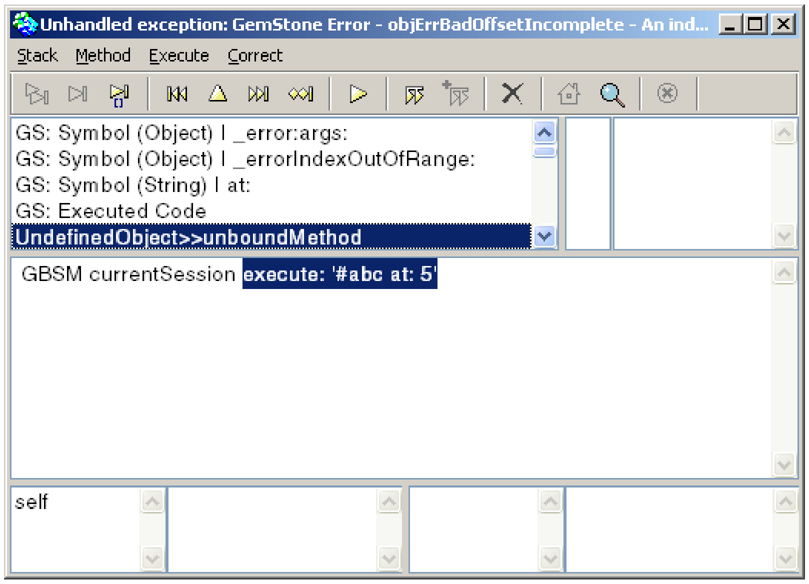
Step 1. In order to reach a debugger, we must first invoke an error notifier. Open a workspace if one is not already open. In it, use the menu item **Do it** to execute the following code:

```
GBSM currentSession execute: '#abc at: 5'
```

Step 2. In the resulting error notifier, click **Debug** to invoke the debugger.

Step 3. The debugger appears. Notice that the top contexts are GemStone Smalltalk contexts (you can see the prepended string `GS:`). Scroll down the messages on the stack until you come to `GS: Executed Code`. Select it to examine it; it is the seam between the GemStone and Smalltalk worlds. The next contexts below are client Smalltalk contexts. Scroll down to see the client Smalltalk code you executed. When you are finished exploring, close the debugger.

FIGURE 1.7 GemStone Debugger


Unhandled exception: GemStone Error - objErrBadOffsetIncomplete - An ind...
Stack
Method
Execute
Correct
self

```
GS: Symbol (Object) | _error:args:
GS: Symbol (Object) | _errorIndexOutOfRange:
GS: Symbol (String) | at:
GS: Executed Code
UndefinedObject>>unboundMethod
```

```
GBSM currentSession execute: '#abc at: 5'
```

Step 4. Open a GemStone Classes Browser to the class GemStoneAdministrator (in the symbol dictionary UserClasses), message category `backup`, message `backupServer: toDirectory:`. (Use the operate button menu item **find class...** if you need to.)

You can invoke a debugger in client Smalltalk by inserting the expression `self halt` in a method. The GemStone Smalltalk equivalent is the expression `self pause`. However, modifying source code in this manner can be time-consuming and error-prone; you may not even be authorized to do so. For this reason, GemStone allows you to set breakpoints in another way.

Step 5. In the text pane of the GemStone Classes Browser, place the text cursor on the line above the return operator, at the end of the line (immediately to the right of the period), and execute the operate button menu item **set break**. You have now set a breakpoint in the method `GemStoneAdministrator >> backupServer: toDirectory:` when it reaches that message-send. A *breakpoint* halts execution and invokes a debugger when the compiler reaches its particular location in a particular method. The breakpoint is now highlighted to show you exactly where execution will halt

NOTE
The following step assumes that you are working on your own private copy of the repository, and that it is small. If that is not true, the next step will take a lot of time and disk space, and you may prefer to skip it.

Step 6. Again in the workspace, select and use **GS-Do it**. to execute:

```
Administrator backupDevelopmentServer
```

Step 7. In the resulting error notifier, click **Debug** to invoke the debugger.

Step 8. A debugger appears, informing you that you have hit a method breakpoint. Select the top element in the stack, which is the breakpoint.

Step 9. In the code window, inspect the code:

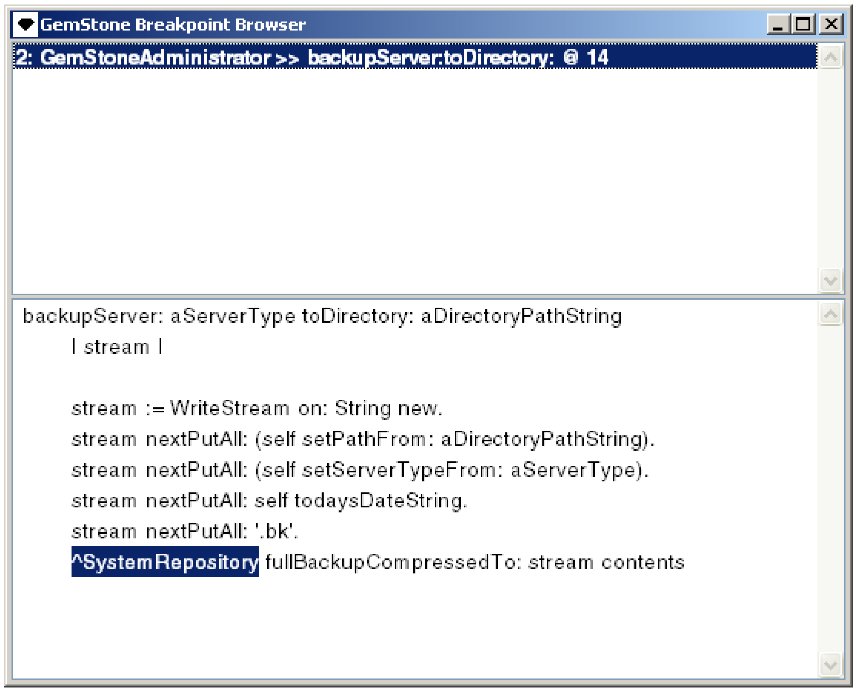
```
stream contents
```

which you'll find at the end of the last line. Click on `self` to see the string constructed to name the backup file.

Step 10. Close the inspector and debugger windows.

Step 11. To remove breakpoints, from the GemStone menu, select **Tools > Breakpoints**.

FIGURE 1.8 GemStone Breakpoint Browser



2: GemStoneAdministrator >> backupServer:toDirectory: @ 14

```
backupServer: aServerType toDirectory: aDirectoryPathString
	| stream |

	stream := WriteStream on: String new.
	stream nextPutAll: (self setPathFrom: aDirectoryPathString).
	stream nextPutAll: (self setServerTypeFrom: aServerType).
	stream nextPutAll: self todaysDateString.
	stream nextPutAll: '.bk'.
	^SystemRepository fullBackupCompressedTo: stream contents
```

Step 12. In the resulting GemStone Breakpoint Browser, the top pane lists breakpoints. Select one, and the associated source code is displayed in the bottom pane, with the breakpoint highlighted. In the breakpoint list, execute the operate button menu item **remove** to clear the selected method breakpoint. (You can also clear all breakpoints at once by executing **remove all**.) Close the Breakpoint Browser when you are done.

NOTE
You can also remove all breakpoints by logging out of the repository. Breakpoints do not persist.

Setting breakpoints and using the breakpoint browser allows you to debug your GemStone Smalltalk code without encumbering it with `self pause` statements.

Step 13. Log out of the repository without committing the transaction.

Step 14. Delete any backup files you've made.

1.7 Summary

As you can see, working in GemStone is in many respects quite similar to working in Smalltalk. However, the two tools have different ultimate purposes, which naturally are reflected by different functionality and behavior under certain circumstances. GemBuilder for Smalltalk is designed to provide as seamless a bridge as possible between the two environments.

The next lesson shows how you can make the temporary objects in your client Smalltalk image become GemStone Smalltalk objects that are permanent residents of the GemStone repository. It also describes the mechanism for ensuring that the two sets of objects remain consistent.

Chapter
2
Persistence

This lesson explores further the interface between the client Smalltalk image and the GemStone repository. You can create objects in the client Smalltalk image, and you can create objects in the GemStone repository. Sometimes, however, you want objects you have created in your image to become GemStone objects that reside permanently in the repository. When a client Smalltalk object becomes a GemStone object, we say that it has become *persistent*.

Once you have created an object in the repository from an application in your image, you often want that object to faithfully reflect the state of its counterpart in the repository. Or you may want an object in the repository to mirror the changes it is undergoing in your client Smalltalk application. When a client Smalltalk object and its GemStone counterpart can be counted on to remain in consistent states, we say that the connection between them has become *transparent*.

This lesson shows you some of the mechanisms that GemBuilder for Smalltalk has implemented for creating and managing object persistence.

2.1 Objectives

When you have finished this lesson, you will:

- know how to manage the connections between Smalltalk objects and GemStone objects, and make Smalltalk objects persistent residents of the repository.

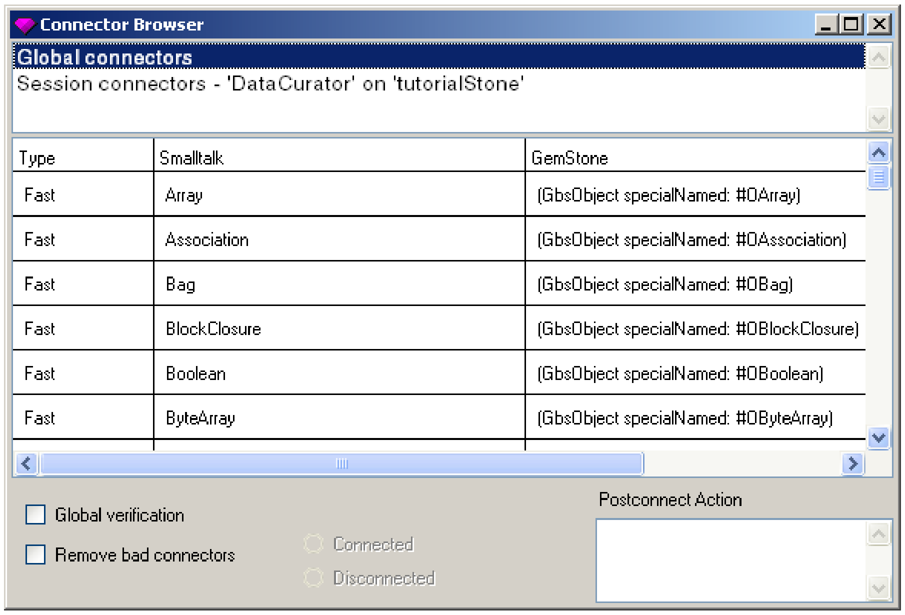
2.2 Connectors

GemStone repository objects can be set up to propagate changes automatically to the corresponding client Smalltalk objects, or vice-versa. However, you must first establish the initial relationship between the GemStone and Smalltalk objects. *Connectors* are the mechanisms we use to establish such relationships. You can create any of several kinds of connectors using a Connector Browser.

Step 1. Log in to your private copy of the repository as DataCurator, if you are not already so logged in.

Step 2. From the GemStone menu, execute **Browse Connectors** to open a Connector Browser. Two sets of connectors are referred to in the top list pane: Global connectors and Session connectors. You can define session connectors for each session whose parameters you have defined.

FIGURE 2.1 GemStone Connector Browser



Global connectors
Session connectors - 'DataCurator' on 'tutorialStone'

Type	Smalltalk	GemStone
Fast	Array	(GbsObject specialNamed: #OArray)
Fast	Association	(GbsObject specialNamed: #OAssociation)
Fast	Bag	(GbsObject specialNamed: #OBag)
Fast	BlockClosure	(GbsObject specialNamed: #OBlockClosure)
Fast	Boolean	(GbsObject specialNamed: #OBoolean)
Fast	ByteArray	(GbsObject specialNamed: #OByteArray)

☐ Global verification
☐ Remove bad connectors

Step 3. Click on Global connectors to select it. A large list of connectors appears in the middle list pane; they connect the GemStone Smalltalk kernel classes to their client Smalltalk counterparts. They are all of type “Fast.” *Fast connectors* connect an object in client Smalltalk to an object identifier in GemStone. If the GemStone object is renamed or redefined, a fast connector continues to point to the old object—the one with the same object identifier.

Four other types of connectors are also available: names, classes, class variables, and class instance variables can all be connected. A *name connector* connects client Smalltalk and GemStone objects by name. Unlike a fast connector, if one of them is redefined, the connector will point to the new object, as long as it has the same name.

A *class connector* connects two classes, verifying that they have the same storage type and instance variables.

A *class variable connector* connects two class variables (shared variables in VisualWorks).

A *class instance variable connector* connects two class instance variables.

Step 4. Select Session connectors.

Step 5. To make a new connector, put the cursor in the middle pane and execute the operate button menu item **add...**.

Step 6. A dialog appears, asking you to specify the type of connector. Select `Name` and click on **OK**.

Step 7. Another dialog asks you to name the Smalltalk object. Enter `STRainbow`.

Step 8. Another dialog asks you to name the GemStone object. Enter `GSRainbow`.

Step 9. Finally, you are prompted for the name of the dictionary in which to place the GemStone object. Accept the default value of `UserGlobals`.

Step 10. Under "Post Connect Action" in the lower right, select **update GS**. The postconnect action defines which side of the connection initially represents valid data and must therefore update the other when the objects are first connected. In this case, **update GS** indicates that the Smalltalk image has the valid data and must update GemStone. If GemStone had the valid data instead, you would select **updateST**.

The postconnect action affects what occurs at login only. After the initial connection is made, changes can propagate in either direction as needed no matter how "Post Connect Action" is specified.

The **forwarder** option creates a Smalltalk object that responds to any message it receives by forwarding the message to the corresponding GemStone object. The **client forwarder** option creates a GemStone object that responds to any message it receives by forwarding the message to the corresponding client Smalltalk object.

Step 11. Open a workspace, if one is not already open. Type the following Smalltalk expression, select it, and execute with **Do it**.

```
Smalltalk at: #STRainbow put: (Array new: 7).
```

Step 12. Now execute with **Do it**:

```
STRainbow at: 1 put: 'red'; at: 3 put: 'green'; at: 6: put: 'indigo';
yourself
```

Step 13. From the top pane of the Connector Browser, execute the operate button menu item **update** to ensure that your workspace changes are reflected in the Connector Browser. Then select the new connector in the middle pane and execute the operator button menu item **inspect ST**. A Smalltalk inspector appears.

Step 14. Return to the Connector Browser and click on the **Connected** button in the bottom pane to connect the connector you just made.

Step 15. Select `GSRainbow` and execute the middle button menu item **inspect GS**. A GemStone inspector appears.

Step 16. In the client Smalltalk array inspector, select the elements tab to assure yourself that the values you have placed there are indeed there.

Step 17. In the GemStone array inspector, select the instance variables to see that they have the same values as in `STRainbow`. The act of inspecting the object has flushed the Smalltalk objects and their values into GemStone.

Step 18. Now type the following in the workspace and execute **Do it**:

```
STRainbow at: 2 put: 'orange'; at: 3 put: 'yellow'; yourself.
STRainbow markDirty
```

Step 19. In the left pane of the GemStone inspector, execute **update**. You can see that the values of the GemStone object have changed correspondingly.

Step 20. Now let's see GemStone's automatic updating in action. In the workspace, type the following; select it and execute **GS-Do it**:

```
GSRainbow at: 4 put: 'green'; at: 5 put: 'blue'; at: 7 put: 'violet'
```

Step 21. Return to the Smalltalk inspector and inspect its instance variables. As you can see, they have the correct values already—the mark dirty mechanism is automatically invoked on the GemStone side.

Step 22. Close both inspectors.

Step 23. In the Session Browser, select the current session if it is not already selected, and click on **Commit...** to commit the current transaction. Answer **yes** to the confirmer.

Step 24. Then click on **Logout...** to log out from the repository and end the session. Answer **no** to the confirmer.

Step 25. In the Connector Browser, select the connector you just made, if necessary, and change the postconnect action to **update ST**.

Step 26. In the workspace, type the following and execute it with **Do it**:

```
Smalltalk at: #STRainbow put: nil
```

Step 27. In the Connector Browser, execute **inspect ST** again, and click on the instance variable in the resulting Smalltalk inspector to assure yourself that the object is indeed nil. Close the inspector.

Step 28. Log back into the repository.

Step 29. Execute **inspect ST** on STRainbow again. This time, it is the old Array object and has all its former values. That's because the login operation connected `STRainbow` and `GSRainbow` and, having clicked **update ST** as the postconnect action, the Smalltalk side was updated by the GemStone object that you committed. Close the inspector.

Step 30. In the Connector Browser, click **Disconnected** to disconnect the two objects.

Step 31. Now click **forwarder**.

Step 32. Click **Connected** again to reconnect them.

Step 33. Still in the Connector Browser, execute **inspect ST** once more. This time, a GemStone inspector appears on `GSRainbow`, because the message `inspect` received by the Smalltalk object was simply relayed to the corresponding GemStone object, which in this case was `GSRainbow`.

Step 34. Close the inspectors and the Connector Browser.

2.3 Summary

This lesson should give you a feeling for the way in which you can manage the problem of updating corresponding objects virtually simultaneously in both the Smalltalk world of your application and the GemStone repository world. You will no doubt grow more sophisticated in your use of connectors as you gain more experience.

In the next lesson, we will set up the example application in both client Smalltalk and in GemStone/S, and we will create the application's users.

Chapter

3

User Profiles and Symbol Lists

In this lesson, we will set up and explore the example application—the Personal Information Manager described in the Introduction—and we will add user profiles for two users, making sure each can access the required symbol dictionary.

Each GemStone user has a symbol list—a list of dictionaries—which together define a separate name space, and hence define the objects that you can refer to in your applications.

Each object you wish to use in your application has a name—a symbol by which you can refer to it. You can define such an object as a temporary variable in a method, for example, but then it is accessible only to that method, and only while that method is executing. If you wish objects to be more generally accessible, you must take steps to make them so.

If you wish other GemStone users also to be able to access an object, then:

- its symbol must be included in a symbol dictionary, and
- all the users who wish to access the object must include that symbol dictionary in their symbol lists.

When you add a symbol to a symbol dictionary shared by other users, the object named by the symbol is accessible to them all.

In practice, of course, most symbol dictionaries include more than one symbol. A symbol dictionary is, in fact, a handy way to collect all the objects defined for a specific application. Used in this way, including a specific symbol dictionary in your symbol list means that you are developing or using the application whose objects are named in that dictionary.

In addition to symbol dictionaries that you create, GemStone users start out with three other symbol dictionaries by default. The dictionary Globals includes references to all the kernel classes and other global objects, thereby allowing all GemStone users to create instances of the kernel classes and perform other standard GemStone operations. The dictionary UserGlobals is available for you to refer to objects that you create for your own purposes. The dictionary Published is for those objects you may wish to make accessible to groups.

In addition, as you saw in a previous lesson, when you define new classes, they appear by default in a newly created symbol dictionary called UserClasses.

3.1 Objectives

When you have finished this lesson, you will:

- understand the general structure of the example application, and how to set up its schema in GemStone/S;
- understand the purpose and components of a user profile;
- know how and why to put symbols into symbol lists; and
- understand the concept of an application's root object.

3.2 Set Up and Explore the Tutorial Application

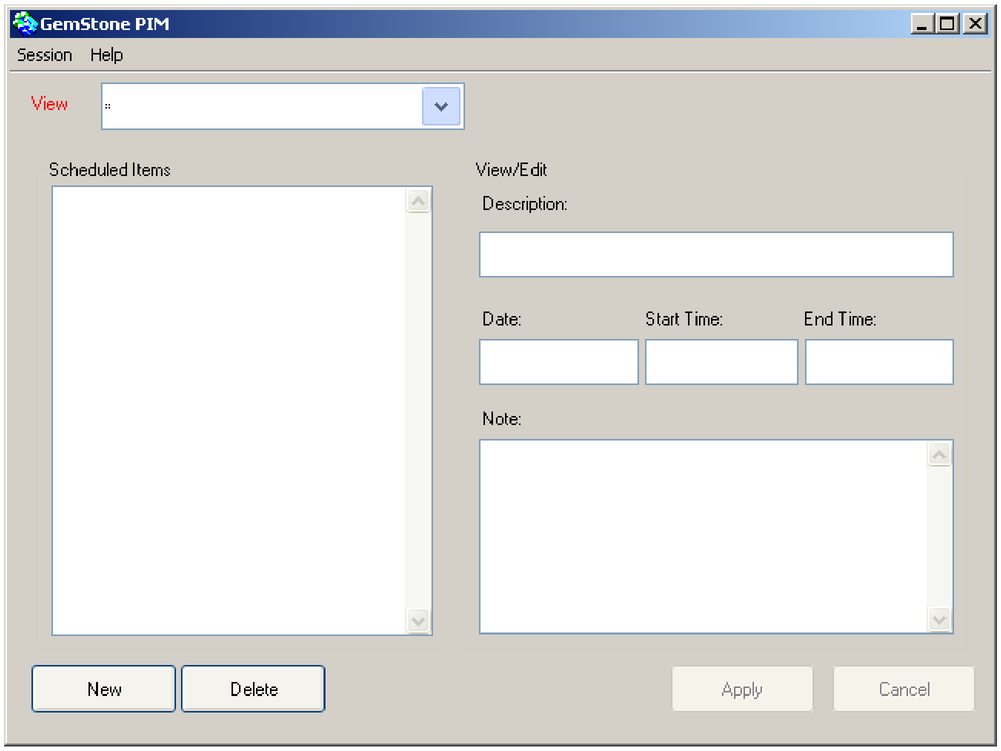
In this exercise, we parcel in the tutorial application and perform some operations to set up the schema classes properly in GemStone.

Step 1. Log into your private copy of the database as DataCurator, if you are not already so logged in.

Step 2. Open a Parcel Browser on the *tutorial* directory.

Step 3. Load the parcel *GsPim.pcl* into the tutorial image. The Personal Information Manager window opens automatically, as shown in Figure 3.1:

FIGURE 3.1 Personal Information Manager Initial Window



Step 4. Save your image.

Step 5. If you accidentally close this window and need to reopen it, execute the client Smalltalk expression:

```
PimManagerUI open
```

Step 6. Take a few moments to explore the tutorial application. Notice the two menus—**Session** and **Help**. Look at them if you wish, but don't execute any of the **Session** menu items until you're instructed to do so, or the rest of this tutorial will not work as expected.

NOTE
To ensure that this tutorial works as described, don't execute any of the **Session** *menu items until instructed.*

Step 7. The top left pane, labeled **View**, starts out with an asterisk in it. Use the drop-down menu to access the **Calendar**.

Step 8. The calendar is initially empty, of course. Click the button labeled **New** at the bottom left.

Step 9. In the entry pane above, select the resulting object, whose text reads: **< Entry is empty >**.

Step 10. Now, in the right side of the window, add a description of your calendar item.

Step 11. In the date and time entry fields, enter the date of the event, its start and end times.

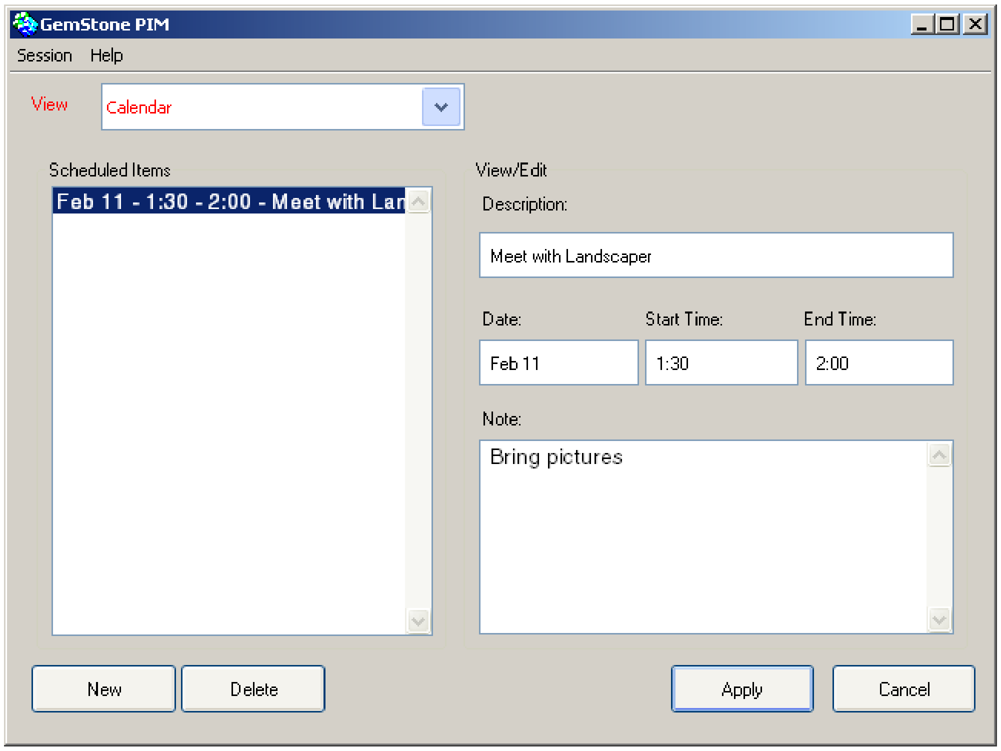
NOTE
The date, start, and end times are not instances of Smalltalk Date or TimeStamp, but simply strings. You can use any format you like to enter this information.

Step 12. If there's anything else you need to remember about this calendar item, enter it in the **Note:** field.

Step 13. When you're done, click **Apply**. The updated item appears in the list at left.

When you're done, your window appears similar to that shown in Figure 3.2. Enter more calendar items, if you wish.

FIGURE 3.2 Calendar Item Added



Feb 11 - 1:30 - 2:00 - Meet with Lan

Description: Meet with Landscaper

Date: Feb 11

Start Time: 1:30

End Time: 2:00

Note: Bring pictures

Step 14. Return to the View field and use the drop-down menu to access the **To Do List**.

Step 15. The To Do List is also empty at first. Click the button labeled **New** at the bottom left.

Step 16. In the entry pane above, select the resulting object, the string which reads: **< Entry is empty >**.

Step 17. Now, in the right side of the window, add a description of your to-do item.

Step 18. In the due date field, enter the date by which the item must be accomplished.

NOTE
This date is not an instance of Smalltalk Date, but simply a string. You can use any format you like to enter this information.

Step 19. If there's anything else you need to remember about this calendar item, enter it in the **Note:** field.

Step 20. When you're done, click **Apply**. The updated item appears in the list at left.

Enter more to-do items, if you wish.

Step 21. Return to the View field and use the drop-down menu to access the **Contacts** list.

Step 22. The Contacts list is also empty at first. Click the button labeled **New** at the bottom left.

Step 23. In the entry pane above, select the resulting object, whose string reads: **< Entry is empty >**.

Step 24. Now, in the right side of the window, add the appropriate contact information.

Step 25. When you're done, click **Apply**. The new item appears in the list at left.

Enter more contacts, if you wish.

Step 26. When you've finished exploring the application, browse the classes that define the model to become somewhat familiar with the code. From the client Smalltalk launcher, execute **Browse > System**.

Step 27. In the Parcel pane of the resulting browser, scroll down till you come to the parcel "GsPim". Select this parcel; you will see a list of the PIM classes.

The classes can be divided into two categories—those that define the application model, and those that define the user interface. The user interface classes are not destined to become persistent—the GemStone/S repository has no graphical user interface, relying instead on a specific interface language and environment (in this case, GemBuilder for Smalltalk) to define and manage an application's user interface. Only the model classes represent the database schema; therefore, only the model classes will be created and compiled in Gemstone/S.

Step 28. We're going to persist the model classes in GemStone, so take a minute to explore them now and familiarize yourself with the code. They are:

PimModel	PimItem
PimUserProfile	PimScheduleItem
PimToDoItem	PimContactItem

When you're done, you can close the image (save it only if you wish to save your Personal Information Manager items), or continue on to the next task.

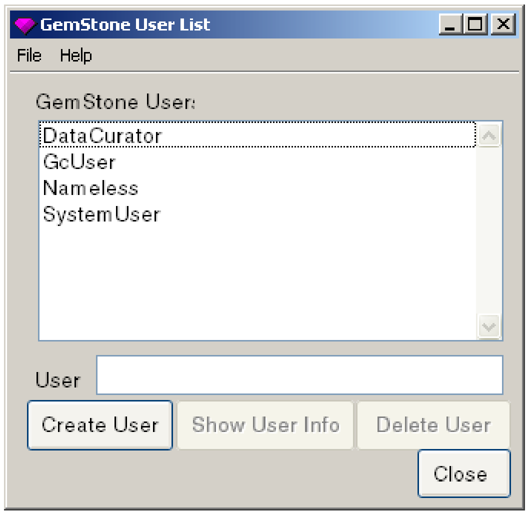
3.3 Exploring User Profiles

GemStone represents its users by means of objects called *user profiles* (instances of the class UserProfile). Each GemStone user has a user profile that sets up his or her username, password, and various other parameters. As DataCurator, you are privileged to create new user profiles (ordinary users are not). We will therefore now create the first user of the Personal Information Manager application—its administrator.

Step 1. Start your tutorial image, if necessary, and log into GemStone as DataCurator.

Step 2. From the GemStone menu, execute **Admin > Users**. The User List appears, as shown in Figure 3.3:

FIGURE 3.3 GemStone User List



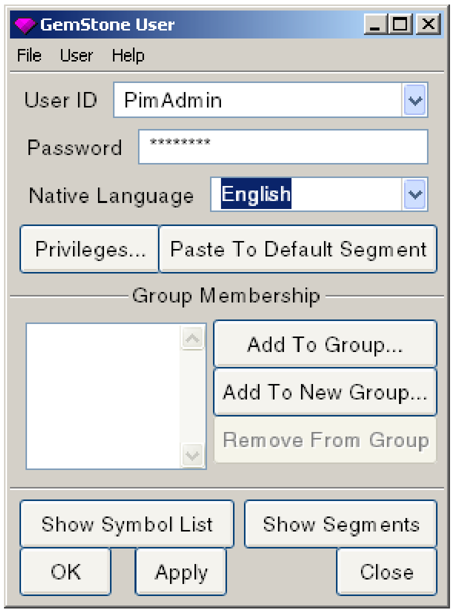
Step 3. Click the bottom button labeled **Create User**.

Step 4. The GemStone User window appears. In the **User ID** field, enter `PimAdmin`. This is the Personal Information Manager user who will serve as the root object for the entire application.

Step 5. Tab to the **Password** field and enter `password`. As the root of the application, this username and password combination is embedded in the code; you'll have more leeway later.

Step 6. Tab to the next field. You'll be prompted for the new password again. Retype the password and click **OK**. The window now appears as shown in Figure 3.4:

FIGURE 3.4 Creating PimAdmin



User ID: PimAdmin
Password: ********
Native Language: English

Step 7. In the bottom half of the window, click the button **Add To New Group...**.

Step 8. When prompted, enter the group name `PimUsers` and click **OK**.

Step 9. Click **Apply** to create the new user.

Step 10. Respond **yes** to the confirmation, and **OK** to the message that changes were saved.

Step 11. Commit the transaction. You can do this from the Session Browser.

Step 12. Back in the GemStone User tool, click **Show Segments**. This invokes the GemStone Segment Tool (which you can also access using the GemStone menu).

A *segment* is a logical entity that allows us to associate *authorizations* with objects. When an object belongs to a certain segment, then the only users authorized to read or modify it are those that can read or modify the segment it belongs to. Therefore, a segment allows us to gather objects that certain users need to be able to manipulate in similar ways. (See Chapter 7, "Object Security and Authorization," in the *GemStone Programming Guide* for further information.)

The segment tool shows us all the segments in the system, their owners (instances of GemStone users), and the types of authorizations—*read* or *write*—that their owners and other users have to the objects assigned to that segment.

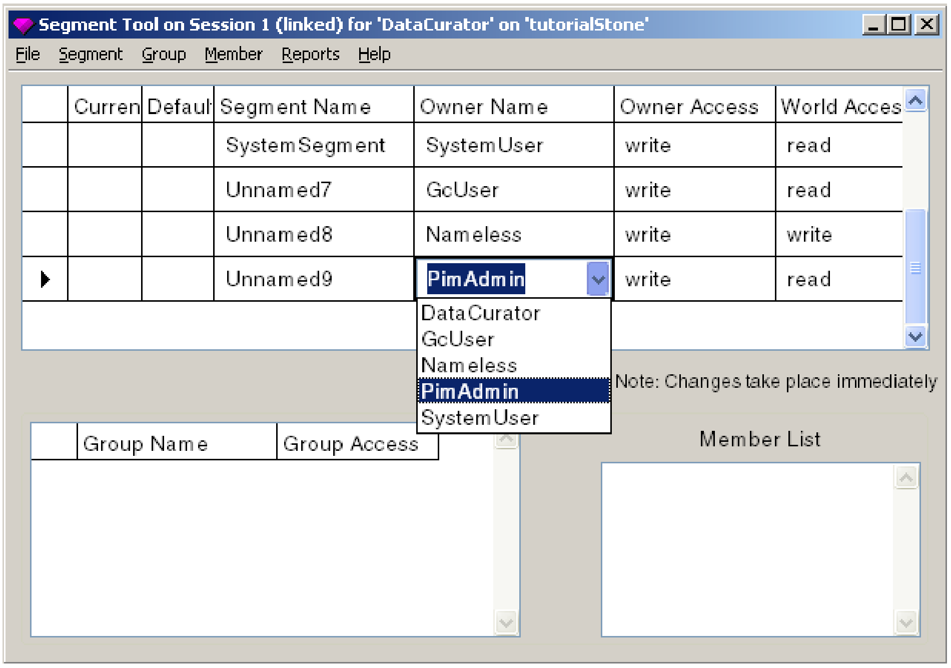
The reason that PimAdmin is the application's administrator is that it is this user's default segment that will hold all the application objects—calendar items, to-do list items, and contacts. When other users add items to their calendars, for example, the new instances will be created in PimAdmin's default segment.

We'll create a group called PimUsers; all users of the Personal Information Manager will belong to this group, and any user in this group will have read and write access to PimAdmin's default segment. In this way, users can freely create and share information with each other without authorization errors, but the application will remain invisible to nonusers.

Step 13. Scroll through the list of segments until you see the one owned by PimAdmin.

Step 14. Select `PimAdmin` in the owner column, and use the drop-down menu to change the segment's owner to DataCurator, as shown in Figure 3.5:

FIGURE 3.5 Segment Tool



	Curren	Defaul	Segment Name	Owner Name	Owner Access	World Acces
			SystemSegment	SystemUser	write	read
			Unnamed7	GcUser	write	read
			Unnamed8	Nameless	write	write
▶			Unnamed9	PimAdmin	write	read

Note: Changes take place immediately

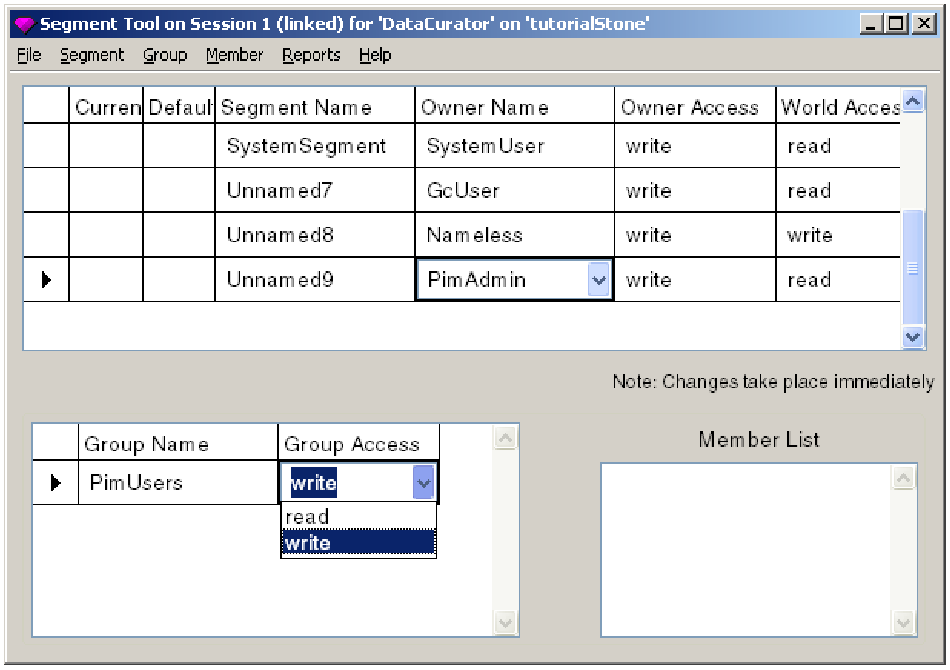
All subsequent users of the Personal Information Manager will also use this segment; if DataCurator owns it and they all have write permission, authorization errors will not occur.

Step 15. In the Groups pane at the bottom left, execute the operate popup menu item **add...**.

Step 16. Add a new group named PimUsers and click OK.

Step 17. Use the drop-down menu to make group access **write**, as shown in Figure 3.6:

FIGURE 3.6 Giving Write Access to the Group


Segment Tool on Session 1 (linked) for 'DataCurator' on 'tutorialStone'
File Segment Group Member Reports Help
Note: Changes take place immediately
Member List

	Curren	Defaul	Segment Name	Owner Name	Owner Access	World Acces
			SystemSegment	SystemUser	write	read
			Unnamed7	GcUser	write	read
			Unnamed8	Nameless	write	write
▶			Unnamed9	PimAdmin	write	read

	Group Name	Group Access
▶	PimUsers	write

The above two steps allow any user belonging to group PimUsers write access to this segment, which DataCurator now owns.

Step 18. Commit the transaction, close the Segment Tool and the User windows.

Step 19. Log out the Data Curator.

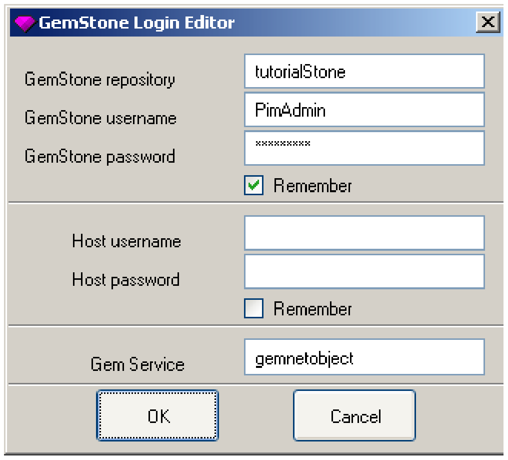
Step 20. In the Session Browser, add a new session. Click **Add...**.

Step 21. In the form that pops up, fill in the name of your private Stone process for the name of the GemStone database. Fill in PimAdmin's user name and password as created above, click on **Remember** (to avoid providing the password each time you log in as this user), and accept the session parameters.

Step 22. Save your image.

Step 23. Select PimAdmin's session parameters in the Session Browser, and log in as PimAdmin, as shown in Figure 3.7:

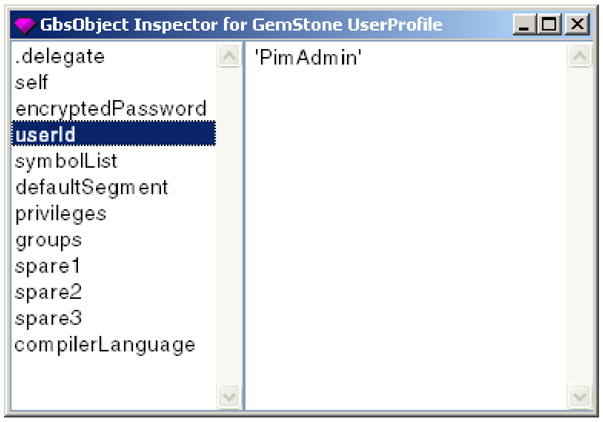
FIGURE 3.7 PimAdmin's Session Parameters



GemStone repository: tutorialStone
GemStone username: PimAdmin
GemStone password: *********
[✓ Remember]
Host username:
Host password:
[ ] Remember
Gem Service: gemnetobject

Step 24. In a workspace, type **`AllUsers userWithId: #PimAdmin`**. Select it and execute **GS-inspect**. An inspector appears on the user profile associated with the GemStone user named PimAdmin that you created above.

FIGURE 3.8 Inspector on a UserProfile



```
.delegate
self
encryptedPassword
userId
symbolList
defaultSegment
privileges
groups
spare1
spare2
spare3
compilerLanguage
```

```
'PimAdmin'
```

The first thing you may notice is that user profiles have a lot of instance variables. Select them each to view their values. Select `userId`, for example, to confirm that you are inspecting the user profile for PimAdmin.

- The `defaultSegment` is the segment into which any objects that PimAdmin creates will be placed, by default. It has associated permissions, as you've already seen.
- The user profile also has instance variables for `privileges`, which in PimAdmin's case are none, and membership in the `group PimUsers`.
- The `compilerLanguage` instance variable allows you to define a language environment in which literal characters or strings in your source code can be compiled into classes other than the default. Such classes can be defined to allow non-European characters or strings.
- The `symbolList` is a list of symbol dictionaries. Inspect it and you will see that it has three elements. Because we have not made any symbol dictionaries accessible to PimAdmin, that account can access only the default three: Globals, UserGlobals, and Published. To see these names, type the following expression into the right pane of the symbol list inspector and evaluate it with **GS-Inspect it**:

```
self collect: [:each | each name]
```

Step 25. Select the instance variables in the resulting inspector to see their values; close the inspector when you are done.

Step 26. When you are finished, close the SymbolList inspector, discarding the text you typed.

Step 27. Now inspect the DataCurator's user profile using the same means.

Three features of the user profile are important to notice:

- By default, each user has a different default segment. However, we have changed this behavior for the Personal Information Manager application, and will assign each user to PimAdmin's default segment.
- No users have special privileges except the DataCurator.
- Each user's symbol list includes the number of elements representing the symbol dictionaries they can access.

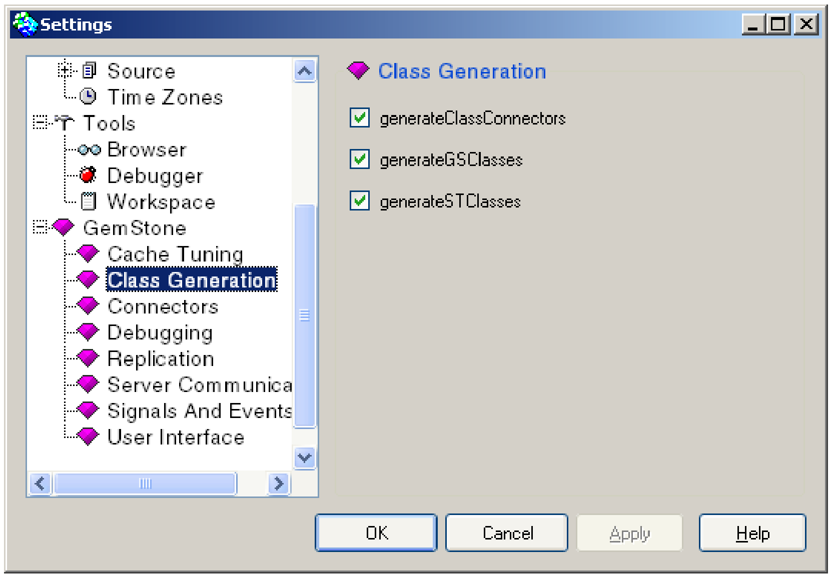
Remain logged into the repository as PimAdmin for the next task.

3.4 Making the Schema Persistent

Now we will create and compile the model classes in GemStone.

Step 1. From the GemStone menu, execute **Tools > Settings**. This invokes a browser on GemBuilder's configuration parameters, allowing you to view them and change their values, if necessary.

FIGURE 3.9 Settings Browser



Class Generation

- [✓] generateClassConnectors
- [✓] generateGSClasses
- [✓] generateSTClasses

The left-hand pane lists all the configuration categories. Select a category, and the parameters in that category are displayed on the right side. Use the **Help** button to open a dialog with the explanation of the parameters. The parameters are displayed and entered in several ways; check boxes, drop down menus of choices, or entry fields.

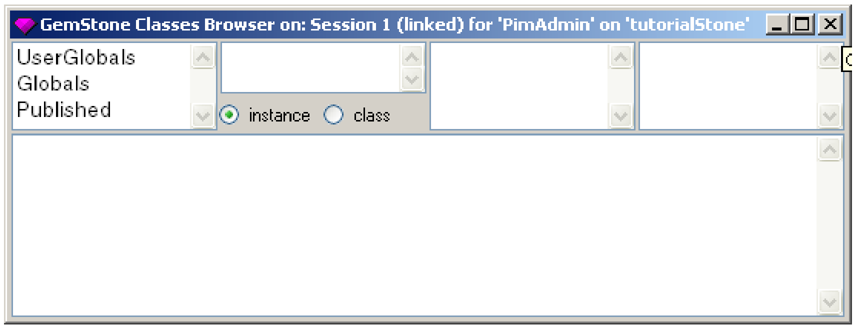
Step 2. On the right, click on the category labeled Class Generation.

Step 3. Verify that the check box labeled generateGSClasses is checked, meaning it is set to `true` (the default). If it isn't, check it, and use the OK button to apply the change.

Step 4. Take this opportunity to read the explanation of this parameter (by using the **Help** button), and to explore the other settings as well. When you are finished exploring, close the Settings tool.

Step 5. From the GemStone menu, execute **Browse > All Classes** to open a GemStone Classes Browser for PimAdmin. As Figure 3.10 illustrates, each browser displays its associated session parameters across its title bar. Because each user has his or her own unique view of the database, a GemStone browser must be associated with a specific session. (You will therefore be opening a new browser for each of your users.)

FIGURE 3.10 GemStone Classes Browser Displays its Associated Session Parameters


GemStone Classes Browser on: Session 1 (linked) for 'PimAdmin' on 'tutorialStone'

Step 6. From the VisualWorks launcher, execute **Browse > System** to open a System Browser. Scroll down till you locate the category PIM-Personal Information Manager, select that, then select the class PimModel.

Make sure the transcript is visible so that you can see the messages GemBuilder writes to it when performing certain activities having to do with replicating classes, which we are about to do.

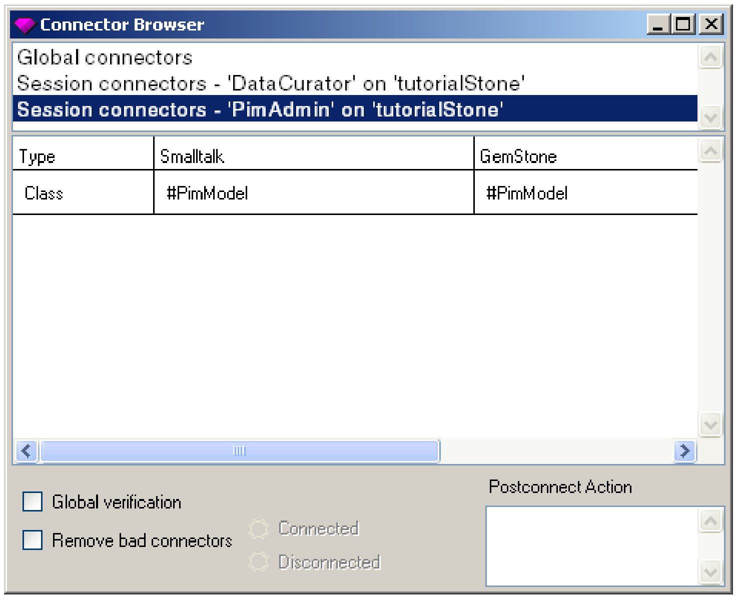
Step 7. With PimModel selected, from the class pane operate popup menu execute **Create in GS**. This creates a corresponding replicate class having the same name in GemStone Smalltalk.

Step 8. From the GemStone menu, execute **Browse Connectors**.

Step 9. In the resulting Connector Browser, select session connectors for PimAdmin. As Figure 3.11 shows, one appears—a class connector for

PimModel, which GemBuilder created automatically as soon as you created the replicate class in GemStone.

FIGURE 3.11 PimModel Connected



Global connectors
Session connectors - 'DataCurator' on 'tutorialStone'
Session connectors - 'PimAdmin' on 'tutorialStone'

Type	Smalltalk	GemStone
Class	#PimModel	#PimModel

Step 10. Return to the System Browser and, using the same menu, execute **Compile in GS**. This compiles all the class's methods in GemStone Smalltalk.

Step 11. For each of PimModel's subclasses, repeat the **Create in GS** and **Compile in GS** steps, checking the Connector Browser as you work if you wish to see each connector as it's created. (Don't forget to update the browser as necessary using the top pane operate popup menu item **update**. Close the Connector Browser and Hierarchy Browser when you're done with it.)

Step 12. Commit the transaction.

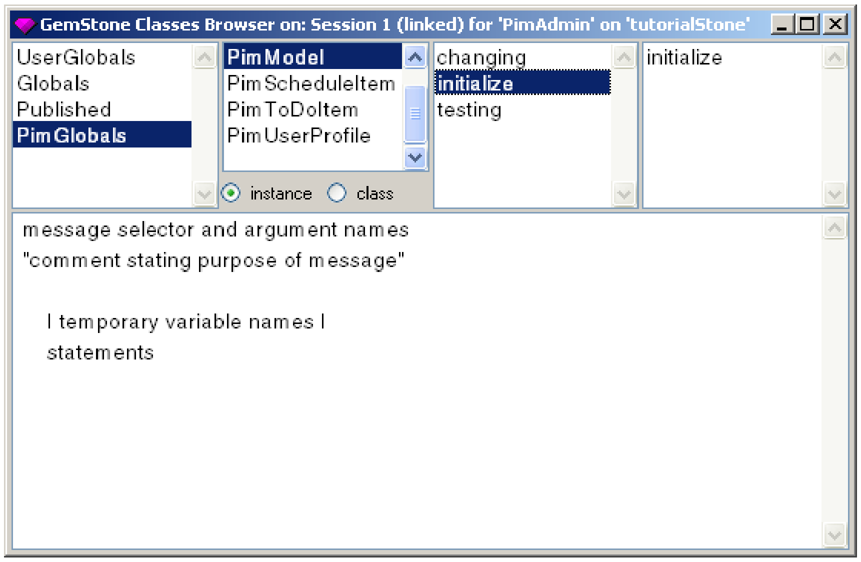
Step 13. Return to your GemStone Classes Browser and browse the symbol dictionary UserClasses to see the classes you just created, and the methods you just compiled.

NOTE
If you had not created UserClasses in a previous task, it would have

been created for you when you created the first GemStone class. Newly created classes are always placed in the symbol dictionary UserClasses, which is created if it doesn't exist. From there, you can freely move classes to other symbol dictionaries.

Step 14. When you've finished browsing, select the symbol dictionary UserClasses in the leftmost pane and execute the operate popup menu item **rename as...**. In the resulting dialog, rename the dictionary PimGlobals, to reflect its true function as the symbol dictionary for the Personal Information Manager's classes. Your browser now appears as in Figure 3.12:

FIGURE 3.12 PimGlobals Symbol Dictionary



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

    | temporary variable names |
    statements
```

If it disturbs you to include the GemStoneAdministrator class in PimGlobals, select that class and execute the operate popup menu item **move to...**. In the resulting dialog, move it to the dictionary UserGlobals.

Step 15. Commit the transaction. Remain logged into the repository as PimAdmin for the next task.

3.5 Initializing the Application

We have now created the user who serves as the application administrator, and persisted the application's model classes in GemStone/S. Our next task is to initialize the application in GemStone/S and create one more user (to stand in for an imaginary host of other users).

We can initialize the application with a single expression that initializes its *root object.* Root objects are important for GemStone/S applications, so let's take a minute to explore the concept. In Smalltalk, an object holds references to other objects—its instance variables and class variables, at least, but often other classes, such as its model. These objects, in turn, refer to other objects, and so on.

The *root object* of an application is the object whose references, when traversed from one to the next all the way through the hierarchy until a dead end is reached, include all the objects required for a given application. The completed graph of such a traversal—all the objects referenced directly or indirectly from a given object—is called a *transitive closure.*

Connectors are often defined on root objects so that, when the application starts, the required hierarchy of objects exists in both the client Smalltalk and GemStone/S.

In the previous lesson, the root object of your "application" was STRainbow, but it wasn't a very interesting root—just a collection with a few instance variables. Nevertheless, when STRainbow was marked dirty, changes occurred to the values of the appropriate instance variables of GSRainbow.

The root object of the Personal Information Manager is the class PimUserProfile's class variable AllUserProfiles: a dictionary whose keys are all the application usernames, and whose values are each user's user profile. Within each user's user profile are instances of the associated calendar items, to-do list items, and contacts, which reference all the application model classes.

We will now initialize the application by initializing its root object.

Step 1. Open a workspace, if one is not already open, and enter the text:

```
PimUserProfile allUserProfiles
```

Step 2. Select it and execute it with **GS-Do it**. This initializes the collection of user profiles for PimUserProfile in GemStone.

Step 3. With the text still selected, execute **GS-Inspect it** to see the (as yet empty) collection of user profiles you've made.

Step 4. Commit the transaction.

Step 5. Log out.

Step 6. Use the Personal Information Manager window's Session menu **Login...** item to log in as PimAdmin. (Click **OK** when you are prompted with the session parameters, and **OK** again to acknowledge the login.)

Step 7. As PimAdmin (the Personal Information Manager title bar shows the user name), add several items to the calendar, to do list, or contact list. Create as many (or as few) items as you wish, but make sure to add at least two items to the calendar.

Step 8. Commit the transaction.

Step 9. From the Personal Information Manager Session menu, execute **Open another window** to get a new Personal Information Manager.

Step 10. From the new Personal Information Manager Session menu, execute **Create new user...**.

Step 11. In the resulting dialog, enter the username PimUser1 and click **OK**.

Step 12. A login editor appears, containing DataCurator's session parameters. Click **OK** to allow the DataCurator (one of two predefined superuser accounts with user creation privileges) to log into GemStone and create the new user.

Step 13. DataCurator now logs in, creates the user, commits the transaction, and logs out. Click OK to dismiss the dialog confirming that PimUser1 has been created with the default password of `password`.

Step 14. If you'd like to see the code that you just executed, open a VisualWorks System Browser and browse the class PimManagerUI, instance protocol `actions`, instance method `createNewPimUser:` as well as the class PimSessionManager, class protocol `initialize-session`, class method `createPimUserNamed: password:`.

Step 15. From the second Personal Information Manager Session menu, execute **Login...**.

Step 16. In the resulting dialog, choose `<New session parameters>` and click **OK**.

Step 17. In the resulting login editor, fill in the username and password for PimUser1 and click **OK**, and **OK** once more to dismiss the confirmation dialog.

Step 18. In PimUser1's Personal Information Manager (make sure to check the title bar), add new calendar items, to-do list items, and contacts as you wish. Be sure to add at least one of each.

Step 19. From the Personal Information Manager Session menu, execute **Commit...** to commit the transaction. Click **OK** to dismiss the confirmation dialog.

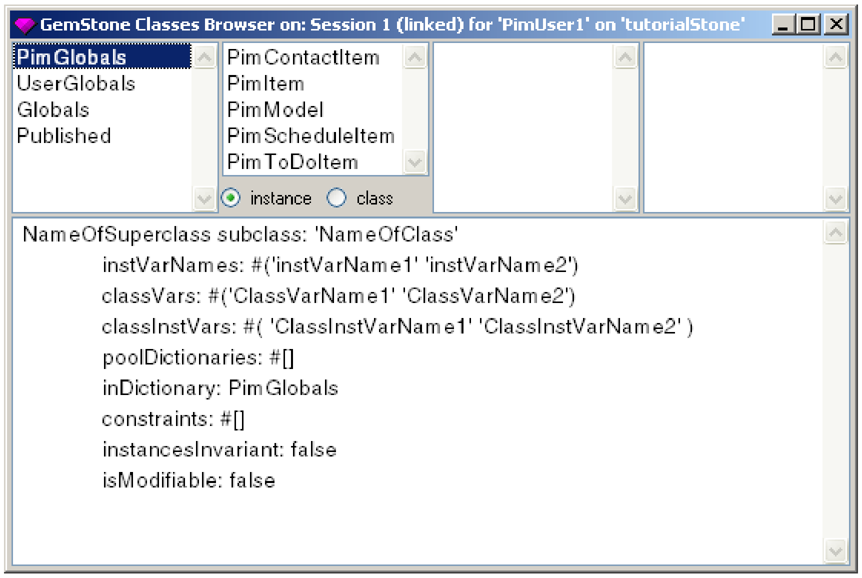
Step 20. From the Personal Information Manager Session menu, log out PimUser1 and close the application window.

Step 21. From the first Personal Information Manager Session menu, log out PimAdmin, but leave the application window open.

Step 22. From the Session Browser, log in as the new user PimUser1.

Step 23. From the GemStone menu, execute **Browse > All Classes**. Notice that, as Figure 3.13 shows, because the new user refers to the same segment as PimAdmin, PimUser1 can also see the PimGlobals symbol dictionary.

FIGURE 3.13 PimUser1 Sees PimGlobals



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

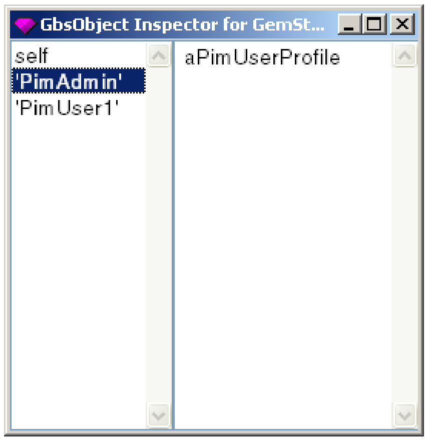
Step 24. Select the symbol dictionary PimGlobals. Notice that, within it, all the Personal Information Manager classes are visible to PimUser1.

Step 25. Select the class PimUserProfile.

Step 26. In the class definition, find the class variable AllUserProfiles and select it.

Step 27. Execute the operate popup menu item **GS-Inspect it**. As Figure 3.14 shows, the resulting inspector includes all the new Personal Information Manager users you've just created.

FIGURE 3.14 AllUserProfiles Inspected



self
'PimAdmin'
'PimUser1'

aPimUserProfile

Step 28. If you wish, select a user in the inspector and inspect it. Continue selecting instance variables and opening fresh inspectors on them until you have reached the items you added to the calendar, to-do list, or list of contacts.

Step 29. When you're finished, log out (no need to commit the transaction).

3.6 Summary

We have now created GemStone Smalltalk classes and instances of the Personal Information Manager model classes, so that each user's calendar, to-do list, and contacts now reside in the GemStone repository. However, while each user has his own persistent data, the information is not available to anyone else. In the next lesson, users will share data.

Chapter 4 Sharing and Querying Data

This lesson explores one of GemStone's mechanisms for querying the database. You can retrieve those elements of collections that meet some specified criterion using ordinary Smalltalk methods for searching collections. However, for collections with thousands of elements, GemStone Smalltalk provides a mechanism that can speed up queries considerably: selection blocks.

A *selection block* is a syntactic variant of an ordinary Smalltalk block designed to optimize queries on large collections. A selection block is in most respects similar to blocks in client Smalltalk, but it is delimited by curly braces `{ }` instead of square brackets `[ ]` and appears as an argument to one of the keywords `select:`, `detect:`, or `reject:`. If you have a collection with more than two thousand elements, querying it using a selection block is faster than querying it using an ordinary block. (For collections with fewer elements, ordinary blocks are probably more efficient.)

(Selection blocks also are usually used on collections that have been indexed and have constrained instance variables. Indexing can also improve the performance of queries over large collections; for details about using selection blocks, indexing, and constraining instance variables, see the chapter entitled "Querying" in the *GemStone Programming Guide*.)

So far, the Personal Information Manager is useful to keep track of your personal schedule, but it cannot help you coordinate appointments with a group. To do that, you'll need to be able to share your personal schedule with other users of the application, and they with you, so that you can see each other's calendars as well as your own.

4.1 Objectives

When you have finished this lesson, you will:

- understand how to share data access among several users, and
- be able to query the database using selection blocks.

4.2 Sharing Data

For this lesson, we will log into the database as PimUser1 and make a few changes to the code in order to share one of the collections—the calendar, or list of schedule items. To do so, you'll have to add an instance variable—*user*—to the class PimScheduleItem, so that each item knows whose schedule it belongs to; you'll also have to make a new class to hold the collection of all users' calendars.

Step 1. Start your image, if it is not already running.

Step 2. Open a Session Browser and log into your private database as PimAdmin if you are not already so logged in.

Step 3. Open a GemStone Class Browser and navigate to the class PimScheduleItem (in the dictionary PimGlobals).

Step 4. Add the instance variable `user` to the class definition and execute **accept**.

Step 5. When you are prompted to migrate instances, answer **yes**. All instances of the class are now instances of its second version, and include this new instance variable with its initial value set to `nil`.

Notice the `[2]` appears after the class name to indicate that the current version is the second.

Step 6. With your cursor in the class pane, execute the operate popup menu item **create access**—a GemStone convenience that makes simple accessing and updating methods for instance variables.

Step 7. You are prompted with an array of instance variables. Cut the variables `startTime` and `endTime`, because we already have accessing and updating methods for these. Leave `#user` in the array and click **OK** to make the methods. They appear in two new categories—Accessing and Updating. browse them and you'll see they are the simplest possible methods.

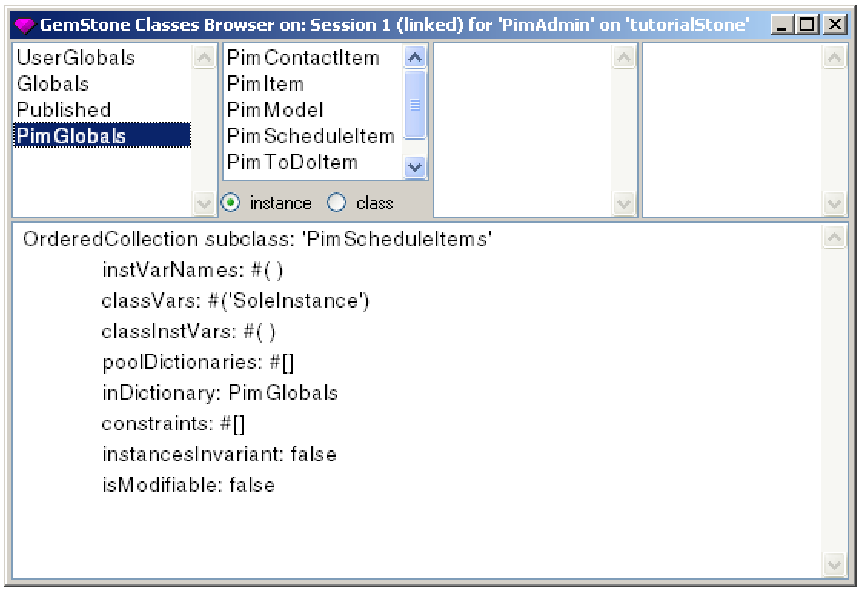
A typical calendar application would track each schedule item with instances of class DateTime; however, we are not constraining the input in any way, and therefore cannot count on anything meaningful on which to sort. We therefore will subclass OrderedCollection and not SortedCollection, as a typical calendar application might.

Step 8. Make a new class PimScheduleItems (with an *s* for the plural). Specify that the superclass is OrderedCollection.

Step 9. Cut all instance variables and class instance variables.

Step 10. Add one class variable named `SoleInstance` to represent the single instance of the calendar and **accept** the definition. When you're finished, the class definition appears as shown below:

FIGURE 4.1 New Class PimScheduleItems



```
OrderedCollection subclass: 'PimScheduleItems'
          instVarNames: #( )
          classVars: #('SoleInstance')
          classInstVars: #( )
          poolDictionaries: #[]
          inDictionary: PimGlobals
          constraints: #[]
          instancesInvariant: false
          isModifiable: false
```

Step 11. Make a new class protocol called accessing, and make a simple accessing class method for `SoleInstance`.

```
soleInstance

  ^SoleInstance
```

Execute **accept**.

Step 12. Make a new class protocol called class initialization, and make a new method called `initialize`, as follows:

```
initialize

  SoleInstance := self new
```

Step 13. Now select the body of the method above and execute it with **GS-Do it** to initialize the new class variable.

Step 14. Commit your transaction.

Step 15. Back in the class pane of the browser, select your new class PimScheduleItems if necessary, and execute the operate popup menu item **create in ST** to create a comparable client Smalltalk class.

Step 16. In the method pane for the `soleInstance` class method—the one that gets the value—execute the operate popup menu item **compile in ST** to create a comparable client Smalltalk method.

NOTE
To ensure that application users cannot accidentally overwrite entire collections of data, compile only this one method to get the value of the class variable and not the whole class. Especially, do not compile the class initialization and class variable setting methods.

Step 17. Open a File List and use **GS-File in** to file in the file `PimUserProfile-converting.gs`. This file contains two new methods to help the conversion from a personal to a shared calendar.

Step 18. Commit the transaction.

Step 19. In a workspace, type:

```
PimUserProfile replaceIndividualSchedulesWithSharedSchedule
```

Step 20. Highlight the above text and execute it with **GS-Do it**.

Step 21. Back in the GemStone Smalltalk Browser, find the class PimUserProfile and its instance method `initialize`. It looks like this:

```
initialize

  super initialize.
  scheduleItems := OrderedCollection new.
  toDoItems := OrderedCollection new.
  contactItems := OrderedCollection new.
```

Step 22. Change the line that initializes the schedule items so that the method reads:

```
initialize

  super initialize.
  scheduleItems := PimScheduleItems soleInstance.
  toDoItems := OrderedCollection new.
  contactItems := OrderedCollection new.
```

Execute **accept**.

Step 23. We've now made all the GemStone Smalltalk changes to implement a shared calendar. Commit your transaction.

Step 24. In a client Smalltalk Browser, access the class definition for PimItemUI and add a new instance variable called `user`. This allows the user interface access to the user attached to each schedule item.

Step 25. Add simple accessing and updating methods for `user`.

Step 26. Now access the class PimScheduleItemEditUI, and its instance method `flushAspects` (in the protocol `updating`) to add a new final line:

```
mdl user: self user
```

This line assigns a user to a given schedule item. Be sure to add a statement separator to the line above, if necessary, and execute **Accept**.

Step 27. Still in the client Smalltalk Browser, access the class PimManagerUI and the instance method `updateView`. We'll change this method, too, so that the calendar subcanvas can be assigned a user.

Step 28. Between these two lines:

```
app model value: self profile.
self mainView: app.
```

add the line:

```
app user: self sessionManager username.
```

and execute **Accept**. The method now reads (with the new line boldfaced):

```
updateView
 "Update subcanvas itemDetailView with the UI for the
currently sekklected view."

 | app viewCls spec subcanvas |
 app := (viewCls := self currentViewClass) new.
 app model value: self profile.
 app user: self sessionManager username.
 self mainView: app.
 spec := viewCls interfaceSpecFor: #windowSpec.
 subcanvas := (self builder componentAt: #mainView) widget.
 subcanvas client: app spec: spec.
```

Step 29. Still in the client Smalltalk Browser, still in the class PimManagerUI, access the instance method `apply` and add a new line after the first line, so that the method now appears as shown below:

```
apply

(self mainView)
  user: self sessionManager username;
  flushAspects;
  refreshItemList
```

Step 30. Finally, we must enable the new schedule item to display its user when printed out in the list. Still in the client browser, access the class PimScheduleItem and give it the new instance variable `user` as well.

Step 31. Make simple accessing and updating instance methods for `user`.

Step 32. In the protocol `printing`, access the instance method `displayFields`, and add `user` to the literal array of fields to be displayed, so that the method reads:

```
displayFields

  ^#( user date startTime endTime description )
```

(Again, the new text to insert is boldfaced.) Execute **accept**.

Step 33. We've now made all the client code changes to implement a shared calendar. Save your image.

Step 34. To see the results of your work, open a new Personal Information Manager. In a workspace, evaluate:

```
PimManagerUI open
```

Step 35. Use the Personal Information Manager's Session menu to log in as PimUser1.

Step 36. Access the calendar, and notice that the schedule items now specify the user to whom they belong.

Step 37. Make new items for PimUser1, if you wish. Then use the Personal Information Manager's Session menu to log in as PimAdmin and make new items for PimAdmin as well, if you wish.

Step 38. In a workspace, type the following GemStone Smalltalk code and execute it using **GS-Inspect it**:

```
PimUserProfile allUserProfiles
```

Open inspectors on each user, their schedule items, and each individual schedule item. You'll see that all the users are now viewing the same collection of calendar items.

Step 39. To make sure, again in a workspace, type the following GemStone Smalltalk code and execute it using **GS-Print it**:

```
(PimUserProfile allUserProfiles at: 'PimAdmin') scheduleItems ==
(PimUserProfile allUserProfiles at: 'PimUser1') scheduleItems
                 true
```

The expression returns `true`; the collections are identical.

Step 40. Close windows and log off, if you wish—or stay logged in for the last task.

4.3 Querying

A Personal Information Manager with just a few users might not require a very large collection of schedule items, but if the application were required to scale to hundreds of users, the calendar could become very large indeed, and performance might become unacceptable.

Also, users might sometimes wish to see only their own calendars, not everyone's.

For these two reasons, we'll now add a query.

Step 1. Start your image, if it is not already running.

Step 2. Open a Session Browser and log into your private database as PimAdmin if you are not already so logged in. (We could actually log in as any Personal Information Manager user, because they all are able to write to the dictionary PimGlobals. But we'll continue to do our development work as PimAdmin.)

Step 3. Open a GemStone Smalltalk Browser, if necessary, and access the class PimScheduleItems.

Step 4. Make a new instance protocol called `querying`.

Step 5. In the new protocol, make the following new instance method:

```
scheduleItemsFor: aUser

  ^self select: {:each | each.user = aUser}
```

Execute **accept**.

NOTE
The above selection block is inefficient for the tiny collection we are accessing; however, selection blocks can greatly improve performance when querying large collections. We are using this one, therefore, to introduce you to this GemStone-specific mechanism.

Step 6. Commit your transaction.

Step 7. Now open a client Smalltalk browser, if necessary, and access the class PimScheduleItemEditUI.

Step 8. Make a new instance protocol called `private`.

Step 9. In the new protocol, make the following new instance method:

```
modelCollection

  ^PimScheduleItems soleInstance
```

Execute **Accept**.

Step 10. Now select the instance protocol accessing, and the instance method `modelList`. Because VisualWorks window widgets require that lists display local rather than forwarded collections, we have to add the message `asOrderedCollection` to the end of the method, so that it appears as shown below:

```
modelList

  ^self model value scheduleItems asOrderedCollection
```

Execute **Accept**.

Step 11. Subclass PimScheduleItemEditUI—call its new subclass PimPersonalScheduleItemEditUI. Give it no instance or class variables; the class definition appears as shown below:

```
Smalltalk defineClass: #PimPersonalScheduleItemEditUI
        superclass: #(PimScheduleItemEditUI)
        indexedType: #none
        private: false
        instancevariableNames: ''
        classInstanceVariableNames: ''
        imports: ''
        category: ''
```

Step 12. The subclass PimPersonalScheduleItemEditUI inherits the instance method `modelList` from its superclass PimScheduleItemEditUI. Make a new instance protocol called `accessing`, and override this method so that the subclass implements it as follows:

```
modelList

  "modelCollection returns either a forwarder if
connected to GemStone, or nil if unconnected. Because the
list widget needs a collection, return an empty one if
not logged in."
^self modelCollection isNil
    ifTrue: [OrderedCollection new]
    ifFalse: [self modelCollection scheduleItemsFor: self user]
```

Execute **Accept**.

Step 13. The previous version of the Personal Information Manager application could count on a one-to-one correspondence between the collection of schedule items held in the GemStone repository, and the collection to be displayed in the user's Personal Information Manager calendar subcanvas. With the addition of the query, however, this one-to-one correspondence is now broken. We must therefore add two new methods to the class PimScheduleItemEditUI to add and remove items from the list displayed in the subcanvas.

To start, therefore, navigate to PimScheduleItemEditUI and make a new instance protocol called `actions`.

Step 14. In the new protocol, make the following new method:

`addNewItem`

```
  | newModel |
  newModel := self modelClass new.
  self modelCollection add: newModel.
  self itemList list add: newModel.
  self refreshItemList
```

Execute **Accept**.

Step 15. In the new protocol, make the following new method:

`removeItem`

```
  self modelCollection remove: self itemList selection.
  self itemList list remove: self itemList selection.
  self refreshItemList.
```

Execute **Accept**.

Step 16. Finally, add the new menu items, so that users can request either their personal calendars only, or the shared calendar.

Step 17. Access the class PimManagerUI, the class protocol `resources`, class method `viewMenu`. Edit the method until it appears as shown below (new text is boldfaced):

```
viewMenu
  "MenuEditor new openOnClass: self andSelector:
#viewMenu"

  <resource: #menu>
  ^#(#Menu #(
      #(#MenuItem
          #rawLabel: 'Personal Calendar'
          #nameKey: #PimPersonalScheduleItemEditUI
          #value: #PimPersonalScheduleItemEditUI )
      #(#MenuItem
          #rawLabel: 'Shared Calendar'
          #nameKey: #PimScheduleItemEditUI
          #value: #PimScheduleItemEditUI )
      #(#MenuItem
          #rawLabel: 'To Do List'
          #nameKey: #PimToDoItemEditUI
          #value: #PimToDoItemEditUI )
      #(#MenuItem
          #rawLabel: 'Contacts'
          #nameKey: #PimContactItemEditUI
          #value: #PimContactItemEditUI ) ) #(4 ) nil )
decodeAsLiteralArray
```

(Don't forget to change the 3 to a 4 in the second-to-last line.) Execute **Accept**.

Step 18. Save your image.

Step 19. Log out of the GemStone Session Browser. (There's no need to commit your transaction, as the work you've just completed all resides in the image.)

Step 20. Open a new Personal Information Manager and log in from the Session menu as PimAdmin.

Step 21. Try the new calendar menu items. You'll see that the Shared Calendar works, but the Personal Calendar is empty. This is because we haven't made a connector for PimScheduleItems's class variable SoleInstance—the sole instance of the shared calendar. Log out and close the PersonalInformation Manager window.

Step 22. Log in as PimAdmin from the Session Browser.

Step 23. From the GemStone menu, execute **Tools > Browse Connectors**.

Step 24. Choose the session connectors for PimAdmin.

Step 25. In the middle pane, execute the operate popup menu item **add...**.

Step 26. When prompted for the connector type, choose Class Variable, because that's what SoleInstance is.

Step 27. When prompted for the Smalltalk class name, enter PimScheduleItems.

Step 28. When prompted for the Smalltalk class variable, enter SoleInstance.

Step 29. When prompted for the GemStone class name, enter PimScheduleItems.

Step 30. When prompted for the GemStone class variable, enter SoleInstance.

Step 31. When prompted for the dictionary, enter PimGlobals.

Step 32. In the bottom right part of the window, choose a postconnect action of type **forwarder**. This means that messages sent to the Smalltalk class variable will be automatically forwarded to the connected GemStone class variable.

Step 33. Finally, in the middle of the bottom, click the radio button labeled **Connected**.

Step 34. Now logout using the Session Browser, open a new Personal Information Manager, and log in from its Session menu as PimAdmin again. Try the Personal and Shared calendars once more. This time you'll see individual and shared calendar items behave correctly.

Unfortunately, having established this connector for PimAdmin, we cannot now log in as the other users without getting a connector conflict—try it and see. Connectors are part of the image, and GemBuilder expects each user to be using his or her own image. This is true for most applications, although it causes awkwardness in tutorials.

If you'd like to be able to log in as the other user, first log out PimAdmin and remove this connector from PimAdmin's session. You are then free to re-create it for the other user's session. Remember, connectors are part of the client Smalltalk image, not part of the repository; they are saved when you save the image and discarded otherwise; they are not themselves persistent GemStone/S objects.

4.4 Conclusion

Congratulations! You have now completed the introductory GemBuilder for Smalltalk tutorial. You have learned the rudiments of navigating GemBuilder for Smalltalk, and you've been introduced to a wide variety of basic GemStone/S concepts:

- persistence by managing the connections between client Smalltalk and GemStone server objects;
- user profiles as representations of users;
- symbol dictionaries for managing the visibility of objects among users;
- groups, segments, and authorization as security mechanisms; and
- specialized selection blocks to speed querying.

You've done a great deal of work and become somewhat familiar with the basic aspects of the GemStone/S system. You are now reasonably prepared to start work on your own application. If you'd like to play with the example application a bit more before starting your own work, we suggest the following possible exercises:

- Allow a user to view a specified user's schedule by itself, instead of with all the calendar items belonging to all users.
- Redefine the model to use instances of Date or TimeStamp (connected to GemStone Smalltalk classes Date and DateTime, respectively) instead of strings for dates and times. Then redefine PimScheduleItems to subclass from SortedCollection.

You probably won't be surprised to discover that you have only skimmed the surface of these concepts, and that GemStone/S includes many more features as well. As you tackle your own tasks, you can investigate further questions as they arise by consulting other GemStone/S documentation.

- The *GemStone Programming Guide* discusses in greater depth the GemStone concepts we have dealt with and others, in a fashion that does not depend on the specific interface to GemStone that you are using. It explains ideas and provides GemStone Smalltalk examples of those ideas.

- The *GemBuilder for Smalltalk User's Guide* for your client Smalltalk describes the GBS in detail, explaining both the user interface and the GBS functionality for accessing the database.
- The *GemStone System Administration Guide* covers system administration issues in thorough detail.
- The *Topaz GemStone Programming Environment* describes Topaz, the command language interface to GemStone/S.
- The *GemBuilder for C* describes the C interface to the GemStone/S repository.
- Finally, on-line help and man pages are available for Topaz, man pages are available for UNIX executables, and the GemStone Smalltalk source code is fully commented.

Happy exploring!

Index