Environment Guide

Contents

Contents

About This Manual

Methods is an object-oriented program development environment for Smalltalk. This manual, the Methods Environment Guide, is a user's guide and reference manual for the environment. Accompanying this manual is the Smalltalk Language Guide, a programmer's guide and reference manual for the Smalltalk language itself.

This manual explains:

- Installing Methods on either a floppy or hard disk (Chapter 1)
- Interacting with Methods to edit text and save your work (Chapter 1)
- Writing Smalltalk code (Chapter 2)
- Using Methods to browse all your files (Chapter 3)
- Browsing, modifying, and extending Methods itself (Chapter 4)
- System maintenance functions (Chapter 5)

Chapter 1: Getting Started

Setting Up Methods

Before proceeding, please take a moment to verify that you have all of the materials. Your Methods package should include the following:

Two Floppy Disks, labeled Source and Image
Methods Environment Guide
Smalltalk Language Guide
Registration Card

The registration card has a serial number on it which matches a corresponding number on the disk labels. The card should be signed and mailed back to:

Digitalk Inc.
5200 West Century Boulevard, Suite 250
Los Angeles, CA 90045

This will enable us to serve you more quickly should you experience any problems.

The floppy disks are not copy protected. Using standard disk copying utilities, you can make one or several backup copies as long as they are for your personal use.

Files

Floppy disk Source contains the Smalltalk source code file, sources.sml. The file is in a compressed format.

Floppy disk Image contains several files:

methods.exe is the machine language program.

image is the Methods environment itself.

change.log is the log in which the system logs all modifications.

go is the file that the system reads when it starts to initialize itself.

sample.sml is a file of sample Smalltalk expressions

misc is a directory of miscellaneous files

There are four pre-configured go files supplied. They are:

color.flp
color.hrd
mono.flp
mono.hrd

You can list their contents since they are normal text files.

The go file contains a series of Smalltalk expressions. The expressions in the go file are evaluated when the system is started. When you learn more about Methods, you can put your own Smalltalk expressions into the go file. This allows you to customize your system as you wish. For instance, you could redefine the keyboard.

The file sample.sml is collection of sample and useful Smalltalk expressions. As you learn more about Methods, you can add to this file as desired.

The directory misc is a collection of miscellaneous files. These include some sample class definitions and the sample user defined primitive. The file readme in this directory explains these files further.

Installation

Make a backup copy of the two disks supplied with Methods. They are not copy protected, so you can use whatever utility you wish. The rest of this section explains how to install Methods after you have made a backup.

For a machine with two floppies, put the disk labeled Image in one of the floppy drives. If your hardware has color and you wish to use it, copy the file color.flp to the file go. You can use the following command:

```
copy  color.flp  go
```

If you have monochrome hardware or if you do not wish to use color, copy the file mono.flp to the file go. The command could be:

```
copy  mono.flp  go
```

For a machine with a hard disk, put the disk labeled Image in the floppy drive. Copy all of the files to the directory in which you wish to install Methods. For example, if you would like to use a directory called methods, the command is as follows:

```
copy   A:\*.*   c:\methods
```

Now put the disk labeled Source in the floppy drive. Copy all of the files to the directory in which you wish to install Methods. You can again use the same command as above.

Go into the directory where you copied all the files. A command which does this is:

```
cd c:\methods
```

If your hardware is equipped for color and you wish to use it, copy the file color.hrd to the file go, as in the command:

```
copy  color.hrd  go
```

If you have monochrome or if you don't wish to use color, copy the file mono.hrd to the file go. A possible command is:

```
copy  mono.hrd  go
```

Running

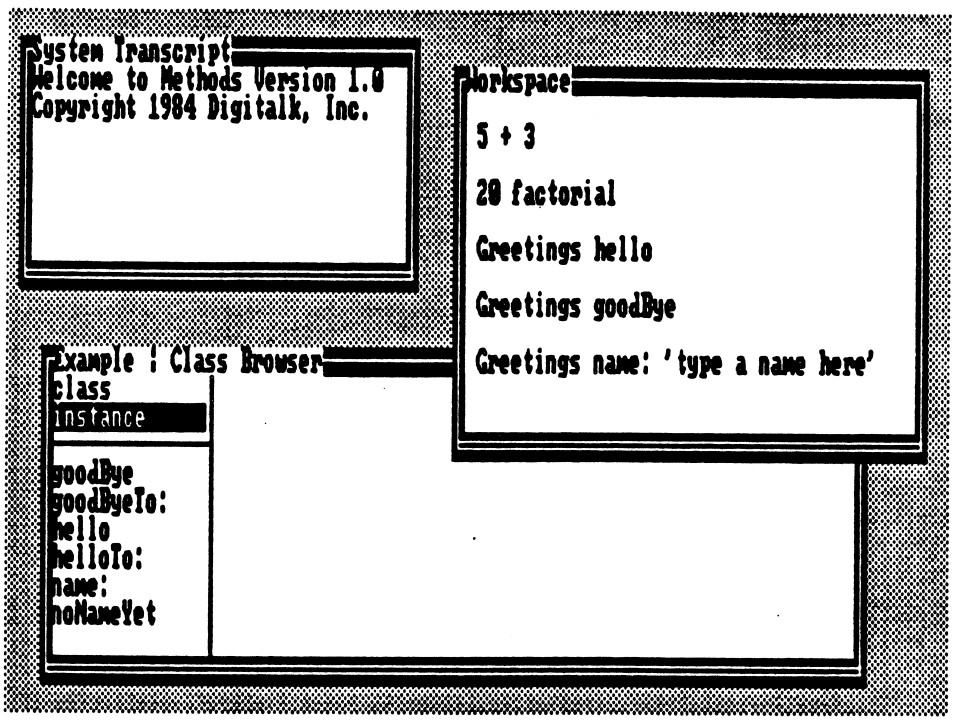
For a floppy disk machine, put the disk labeled Image in drive A and put the disk labeled Source in drive B. Make sure that the current drive is drive A. If not, type A: and then press the return key. Now type the word methods; and press the return key.

For a hard disk machine, enter the directory where you copied all the files. Use a command such as the following:

```
cd c:\methods
```

Then type the word methods and press the return key.

Appearance of Screen

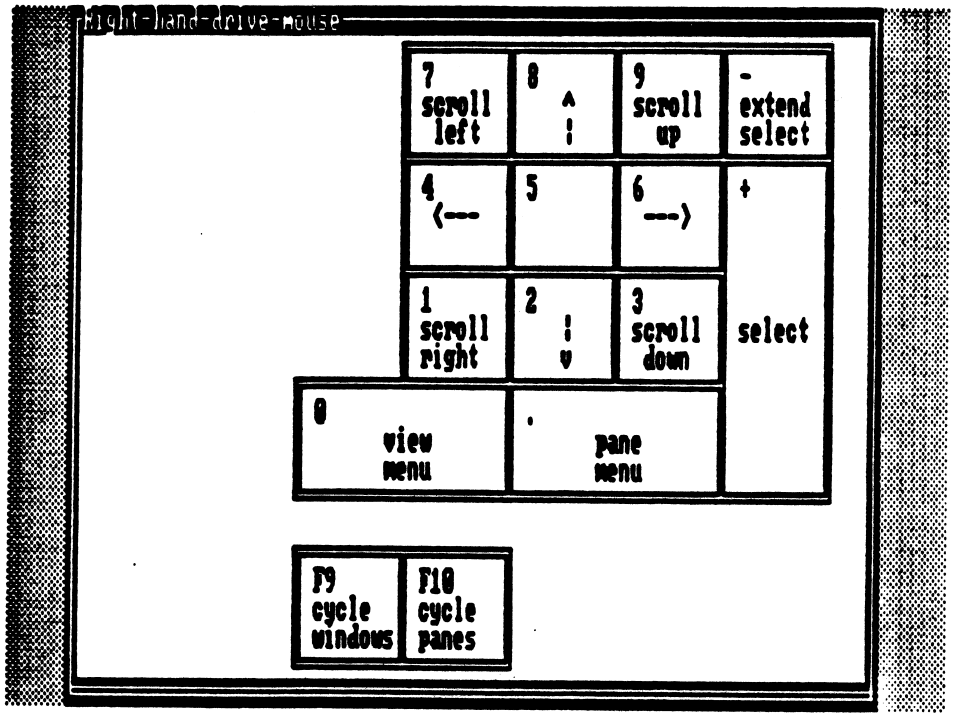


This is a picture of the Methods screen with three windows and is the initial start up screen of the system as it is shipped. Note that if your system is set up for color, the windows may be different colors. In our example, we have a System Transcript, a Workspace, and a Class Browser.

This first chapter will show you how to make these and other windows appear and disappear, how to modify the contents of windows, how to move from window to window, and how to exit and restart the Methods environment.

Every window in Methods has a label and is surrounded by a border. The area inside the border is divided into one or more panes. In our example, the Workspace and the System Transcript each have a single pane, and the Class Browser has three panes.

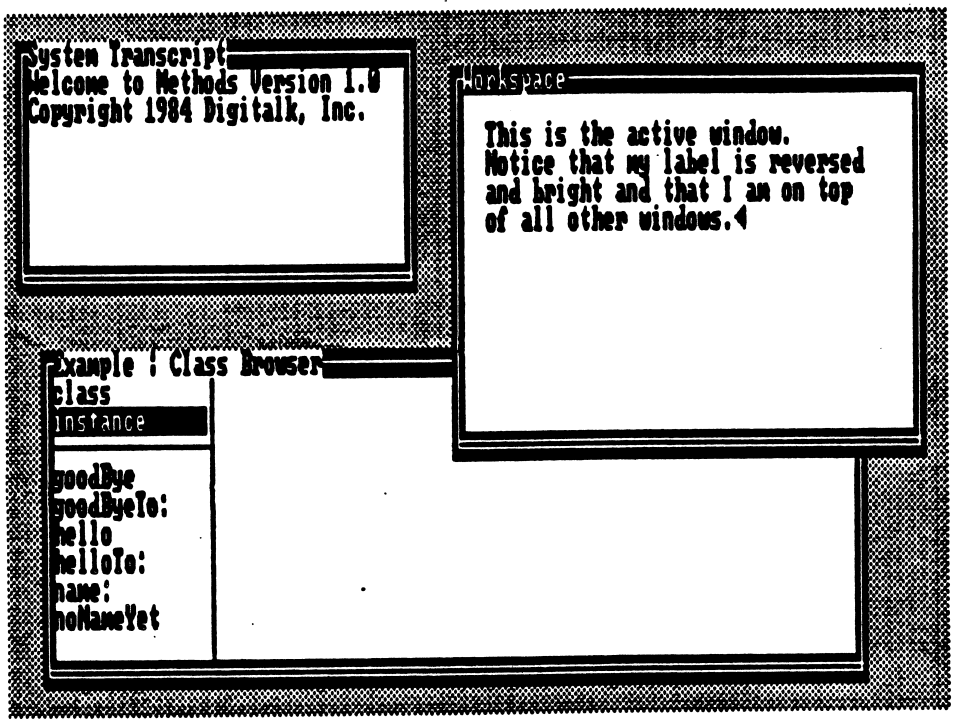
Right-Hand-Drive Mouse



The flashing underline on the screen is the cursor. The cursor is moved by pressing the cursor arrow keys on the keyboard. Pressing an arrow key moves the cursor one space in the direction of the arrow. Pressing the arrow key while holding down the shift key makes the cursor jump several spaces.

The keys surrounding the cursor arrow keys make up what we call the right-hand-drive mouse. The figure above shows the keys that make up this keyboard mouse and the names that we have given them. The other keys on the keyboard are used for typing. The keyboard layout, along with anything else in this environment, can be changed by you.

Active Window



Move the cursor about the screen and notice what happens as it enters and leaves a window.

The window that contains the cursor is the active window and it always has its label reversed and highlighted. If the cursor is outside all of the windows, then there is no active window and all of the labels have a normal appearance. Methods uses many visual clues like this to enable you to always know what is happening.

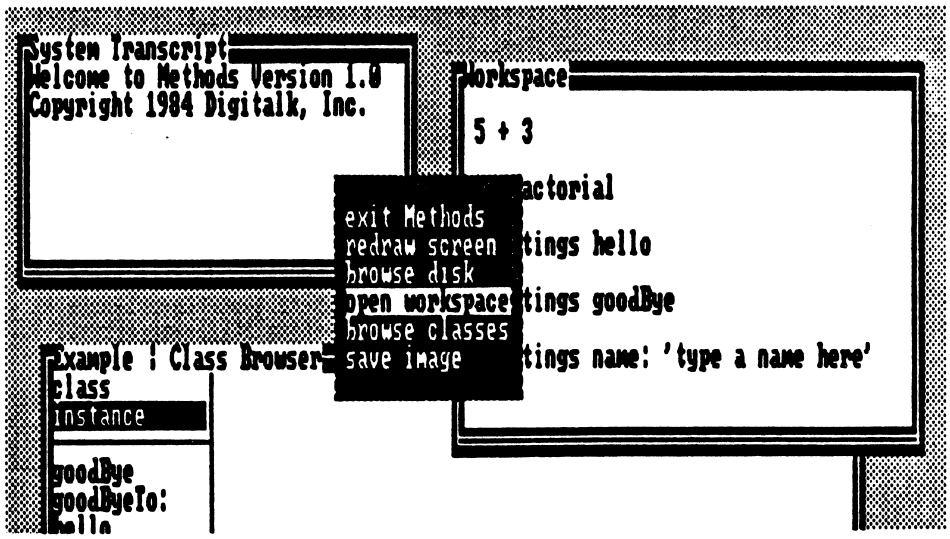
Cycling

Moving the cursor about the screen using only the cursor keys can become laborious; therefore, Methods has a fast way of jumping about the screen. Move the cursor into a window and then press the window cycle key, the 'F9' function key on a standard IBM keyboard. Do this several times and notice what is happening. The cursor is jumping between the windows in a cyclic fashion. This is called window cycling.

Move the cursor into a window with more than one pane. Now press the pane cycle key, the 'F10' function key on a standard IBM keyboard, several times and notice what happens. The cursor is jumping between the panes of the active window in a cyclic fashion. This is called pane cycling.

Using Menus

System Menu



Move the cursor until it is outside of every window on the screen and then press either the window menu key, the 'Ins' key on a standard IBM keyboard, or the pane menu key, the 'Del' key on a standard IBM keyboard. This pops up the system menu, a picture of which is shown above.

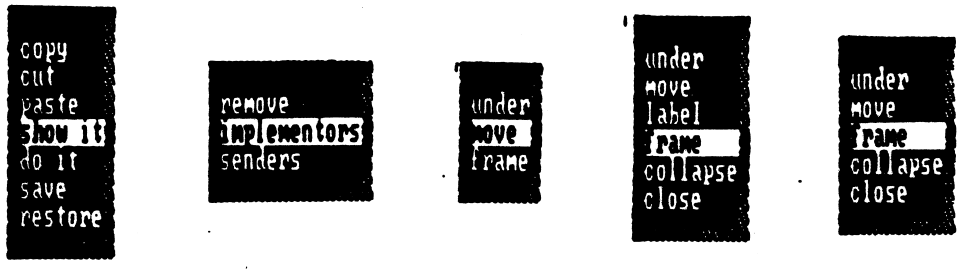
Menus are the standard way of giving commands to Methods. The system menu has commands for creating new windows, saving your work, exiting the environment, redrawing the screen, and coloring the screen background if your system is set up for color. The menu shown above is from a monochrome system and therefore the color screen function is not in it.

Move the cursor up and down. Notice that the line on which the cursor sits is always reversed. To leave a menu without performing any function, move the cursor outside of the menu in any direction. The menu disappears. Notice that the quickest way out is to move the cursor to the left.

To select a menu function, move the cursor on top of the desired function and press the select key, the large plus (+) key to the right of the numeric key pad on a standard IBM keyboard. As an example, pop up the system menu by moving the cursor outside of every window and pressing either menu key ('Del' or 'Ins').

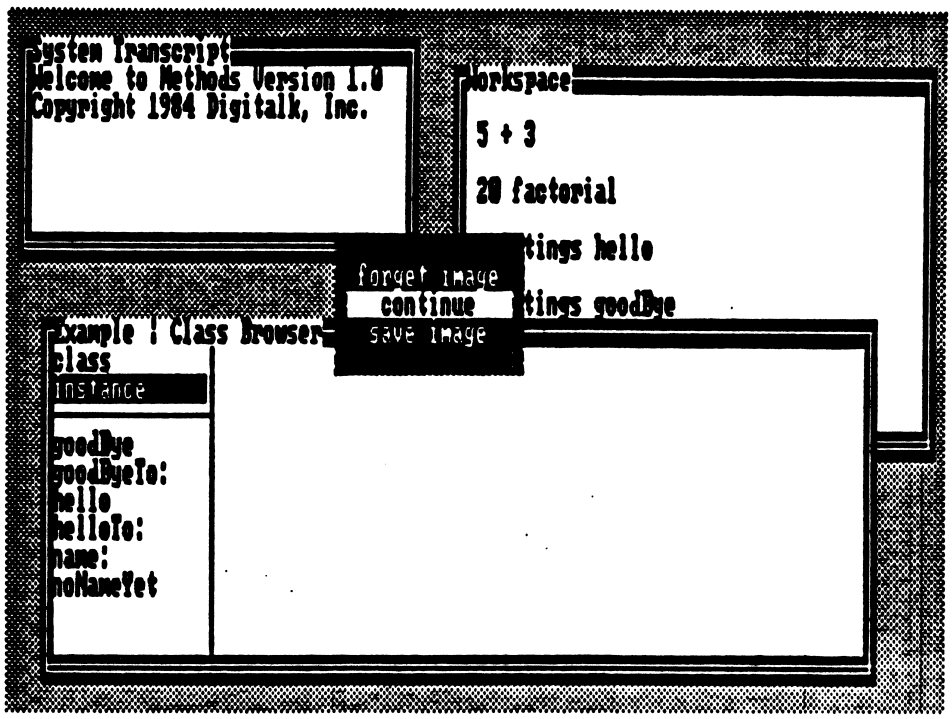
Move the cursor until the redraw screen function is reversed. Now select the redraw screen function by pressing the select key (large + key). The action is immediately performed and the menu disappears.

Window and Pane Menus



There are several menus in the system, some examples of which are shown above. Move the cursor around the screen from pane to pane and window to window (cycling is easiest) and pop up the menus using either menu key. (Remember that you must leave a menu before you can cycle or pop up a different menu). Notice that the menu that you get is determined by the location of the cursor and which menu key you use. When the cursor is inside of a pane and the pane menu 'Ins' key is pressed, a menu specific to that pane is popped up. This is called a pane menu. When the cursor is anywhere inside of a window and the window menu 'Del' key is pressed, a menu specific to the window is popped up. This is called a window menu. When the cursor is outside of every window, the system menu is popped up.

Exiting and Restarting



To exit the methods environment, pop up the system menu (by pressing the menu key when the cursor is outside of every window) and select the exit methods function (by positioning the cursor over "exit methods" and pressing the select key). Methods then responds by popping up another menu with three choices as shown above. Methods wants to know if the changes made since the environment was started are to be remembered or thrown away.

Selecting the save image function causes the state of the system, including the location and contents of all the windows, to be remembered. When Methods is started the next time, it will be restored to the saved state.

Selecting the forget image function causes all of the changes made since Methods was started to be forgotten. When Methods is started the next time, it will be restored to the previous saved state (the way it was when it was started this time).

Selecting the continue function returns you back to the Methods environment. This has the same effect as leaving the menu without selecting anything.

The image is all of the Smalltalk program code and Smalltalk data areas that make up the Methods environment. The image file is the file that the system reads when it starts up. It contains the program and the initialized data structures that are loaded into memory. These initialized data structures include the windows that appear on the screen. Therefore when you start up the system, it resumes exacly as it was when the image file was written.

Since Methods is an interactive and modifiable environment, the code and data are constantly being changed as you use and modify Methods. These modifications are not written on the disk until you ask them to be written, either by selecting the save image function while exiting or by selecting the save image function from the system menu.

The image file contains pointers into the other two main files of Methods, the source file (sources.sml) and the change log (change.log). You must maintain these three files as a group.

Since the image file on the disk represents a guaranteed starting point in the event of a system crash of a major mistake, you should maintain a recent backup of the image file along with the source file and the change log that are associated with it.

Chapter 5 in this manual explains the relationships between these files in more detail, and gives instructions on maintaining them. It also gives some advice on how to recover from a system crash or other disaster.

To restart Methods, use the same procedure as described in the section "Running" above.

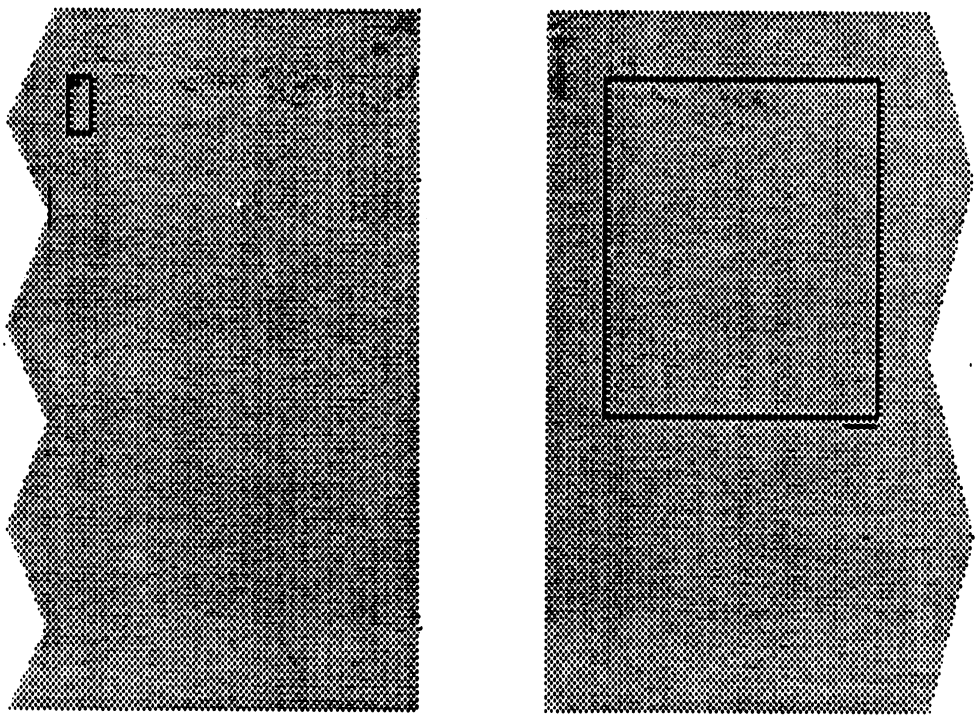
Manipulating Windows

Opening a Workspace

Windows are usually opened by making selections from menus. To open a new Workspace, move the cursor outside of every window, pop up the system menu using the menu key, and then select the open workspace function using the select key.

Methods responds by displaying a blinking rectangle as shown below. This is a visual clue that Methods is waiting for you to define the rectangle the new window is to occupy on the screen.

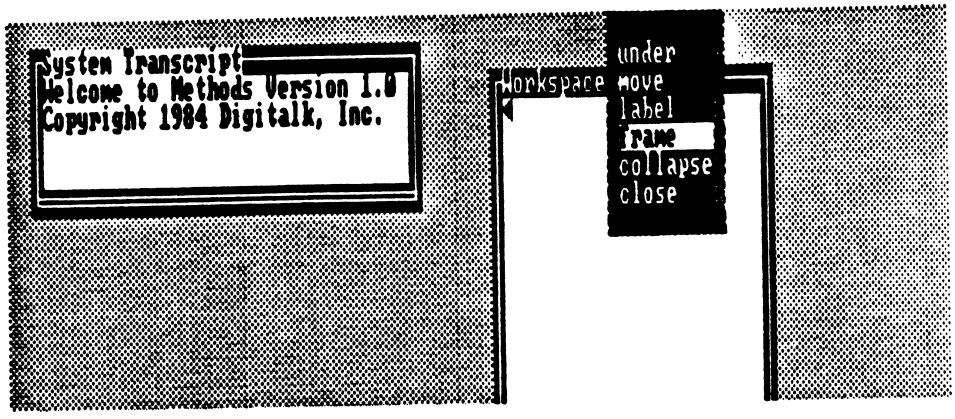
Defining a Rectangle on the Screen



To define the rectangle, move the cursor until the upper left corner of the blinking rectangle is where the upper left corner of the new window is to be. Then press the select key. Notice that the cursor jumps to the lower right corner of the blinking rectangle. Now when the cursor is moved, the blinking rectangle

stretches and shrinks. When the blinking rectangle is the size that the new window is supposed to be, press the select key. The new window pops up.

Common Window Menu Functions



The window menu of every window has functions that apply to the window as a whole. The picture above shows the window menu for a Workspace.

To move a window, pop up the window menu (press the window menu key with the cursor inside of the window) and select the move function. Methods responds with a blinking rectangle that is the size of the window being moved. Move the cursor until the blinking rectangle is in the desired location and press the select key. Methods moves the window to its new location.

To frame a window (move and resize it), pop up the window menu and select the frame function. Methods responds with a blinking rectangle. Move the rectangle until the upper left corner is in the new location and press the select key. Now stretch or shrink the rectangle until it is the proper size and then press the select key. Methods moves the window to its new location and changes its size to the new size.

Collapse causes the window to be collapsed so that just the label is visible. The contents of the window are remembered. To make its contents visible again, frame the collapsed window as explained above.

Under causes the window underneath this window at the current cursor position to pop up on top and become the active window.

Label lets you type in a new label for the window. When label is selected, Methods pops up a Prompter window in which a new label can be typed. Just type the new label and press the return key when finished. Backspacing deletes the last character typed.

Close causes the window and its contents to be thrown away.

If You've Got Color

If your hardware allows for color and you have installed your system for color as explained on page 4, you will have noticed that the system menu and the window menus are slightly different from the menus shown in the previous section. The system menu includes the color screen function and the window menu includes the color function.

Coloring the Screen

To change the color of the screen background, pop up the system menu (by moving the cursor outside of every window and pressing either menu key) and select the color screen function. The color menu with eight bands of color appears. These are your color choices for the screen background. Move the cursor to the color you want and select it. The screen background changes to that hue.

Coloring a Window

If you wish to change the color of a window and/or the text in a window, pop up the window menu and select the color function. Another menu pops up. This menu gives you the choice of coloring the foreground text or the background window.

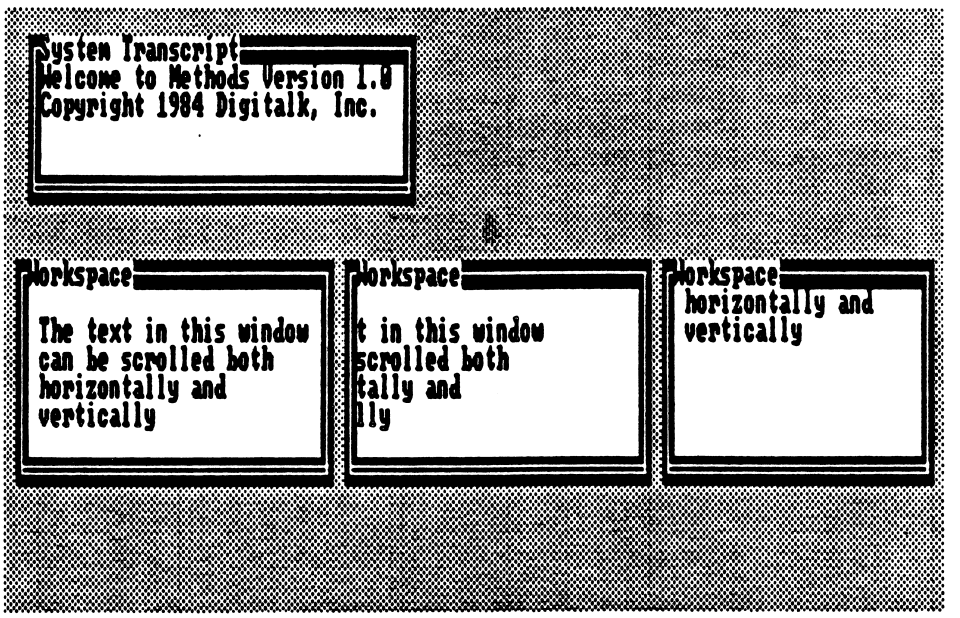
If you choose the color window function on the second menu, all of the panes of that particular window will have their background color changed.

If the color text function is chosen, then the color of all the text in the window will be changed.

After selecting either of the above choices, the color menu pops up and you can select the color.

Setting the text color to the same color as the window color naturally makes the text impossible to see. Bring up the menu again and select an alternate color for either the text or window background and the text becomes visible.

Scrolling



There are many instances when the contents of a pane are larger than the pane. Even though you may not see the text that is outside of a pane, it is still there and scrolling brings it into view. Scrolling moves the pane over the underlying contents so that a different portion of it becomes visible.

Vertical scrolling is accomplished by pressing the Page Up and Page Down keys. Pressing the page up key causes the pane to move up one line. This causes portions of the text nearer the beginning to become visible. Pressing the page down key moves the pane down one line.

Horizontal scrolling is accomplished by pressing the Home and End keys. Pressing the home key moves the pane to the right four characters. Pressing the end key moves the pane left four characters.

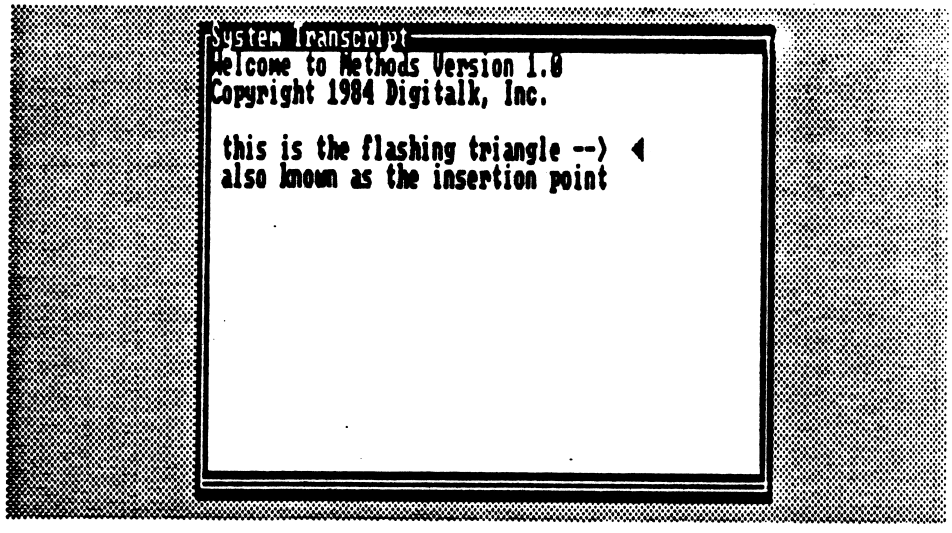
Holding down the shift key when pressing any of the four scroll keys causes the pane to move a larger amount in the indicated direction. For vertical scrolling, the pane moves almost the entire height of the pane. For horizontal scrolling, the pane moves one half of the pane width.

Editing Text

Text Panes

Many of the windows in the Methods environment have panes that allow you to edit text. These are called text panes. The Prompter used in changing the label of a Workspace (described above) is a window with only a single text pane, as are the System Transcript and any Workspace. These and all other text panes use the same text editor.

Inserting Text



Move the cursor into the System Transcript window. For this example, make sure that it is in a blank part of the pane. Now press the select key. A bright flashing triangle appears. This triangle is the text insertion point. Now type some characters. Notice that the characters are inserted in front of the flashing triangle. If the backspace key is pressed, the character in front of the insertion point is deleted.

To move the insertion point, place the cursor at the new position and press the select key. The insertion point jumps to the cursor position. Notice what happens if the cursor is placed over a character and the select key is pressed. The character starts to flash. This way of indicating the insertion point avoids obscuring the character.

Move the cursor into the border or label of the System Transcript. Notice that the flashing insertion point disappears. This is a visual clue that the cursor is outside of the text editing pane. Now try to type a character. There is a beep and the character does not appear anywhere. You can only edit the contents of a text pane if the cursor is inside of it.

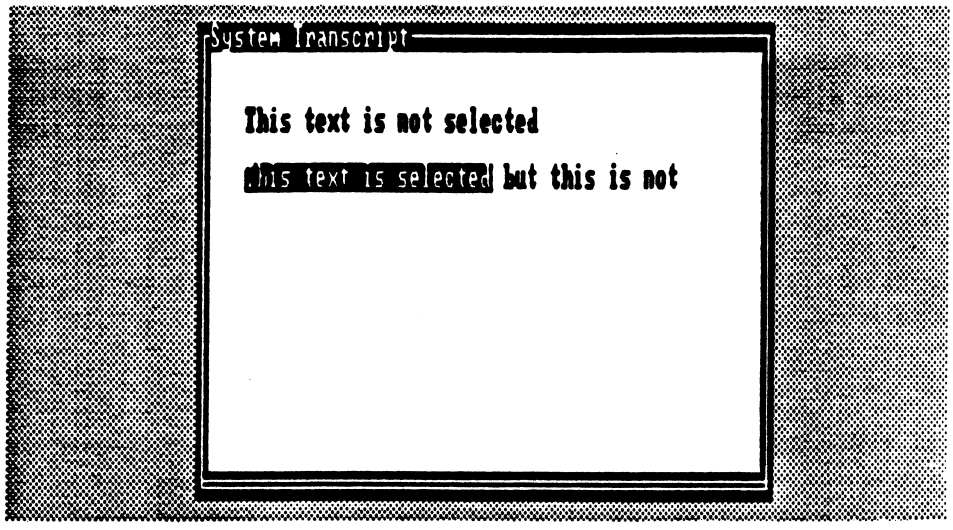
Return, Tab, and Backspace Keys

Move the cursor inside of the System Transcript and press the return key. Notice that a new line is inserted and the insertion point is placed at the start of this new line. If you press the return key when the insertion point is in the middle of a line, the line will be split at the insertion point into two lines.

If you press the tab key, blanks will be inserted at the insertion point to make it move to the next tab stop. In this editor, tab stops occur every four spaces.

If you press the backspace key, the character in front of the insertion point is deleted. Notice that if the insertion point is at the beginning of a line and backspace is pressed, the line is joined to the line above.

Selecting, Replacing, and Deleting



To select some text, move the cursor to either the beginning or end of the text to be selected (this can be in the middle of a word) and press the select key. This will bring the insertion point to the cursor. Now move the cursor to the other end of the text to be selected (this can also be in the middle of a word) and press the extend selection key (the '-' key to the right of the numeric keypad on a standard IBM keyboard). The entire selected area will appear in reverse video. Notice what happens if you move the cursor and press the extend selection key again. The selected (reversed) area of text is changed to end at the new cursor position. You can continue to move the cursor and adjust the selected text by pressing the extend selection key until it is exactly as desired.

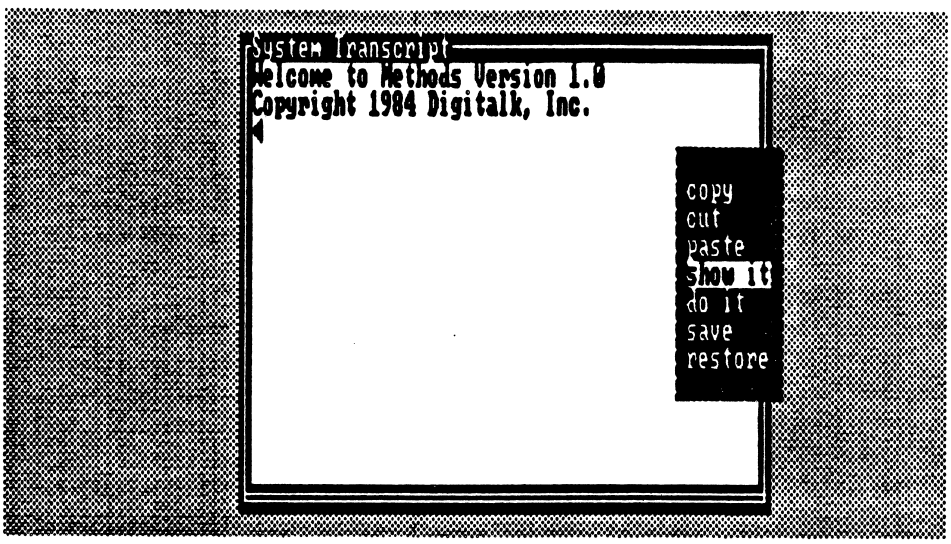
If you press the select key, the selected text is de-selected (no longer reversed) and the insertion point appears at the cursor.

Replacing text is done by selecting the text that is to be replaced and then typing the new text. As you type, Methods replaces the selected text with the new text.

Deleting large amounts of text can be accomplished by selecting the text and then pressing the backspace key. For small amounts of text you can also place the insertion point at the end of the text to be deleted and press the backspace key repeatedly.

Saving and Restoring

The text editor in Methods is always working with a text copy of some underlying object in the system. Among other things this can be a file, a string, or some Smalltalk code. For example, the System Transcript and any Workspace edit strings of characters, and the text pane in the Class Browser edits Smalltalk code. Because a copy is being edited, it is necessary to tell the environment when editing is complete. Methods can then store the edited text back in the original place. It is also possible to restore the text in the pane to the way it was before editing started.



Every pane that allows text editing has editing functions on its pane menu. The picture above shows the pane menu for the System Transcript. (To pop up the menu move the cursor inside the pane and press the pane menu key.)

Move the cursor into the System Transcript and type some characters. Now pop up the pane menu, select the restore function, and notice what happens. When the restore function is selected Methods replaces the text in the pane with the text representation of the underlying object being edited. If a string is being edited (as in this case) then the string replaces the text in the pane. If a file was being edited, the text in

the file is reread into the pane. If Smalltalk code is being edited, then the source code currently being used by the system is put in the pane.

The save function in the pane menu tells Methods that editing is complete. Methods responds by updating the underlying object being edited with the text currently in the pane. For example, if a file is being edited, then the file is rewritten with the text in the pane.

Cutting, Copying, and Pasting

The text editor has a buffer that can be used to transfer text from place to place. Text is placed in this buffer by either cutting or copying it from the pane. After text has been placed in the edit buffer, it can be pasted anywhere inside of a text pane.

To place text in the edit buffer you must first select the text (make it appear reversed). Then pop up the pane menu and select either cut or copy. If the cut function is selected, Methods responds by replacing the contents of the edit buffer with the selected text and then deleting the selected text from the pane. If the copy function is selected, the selected text is not deleted from the pane. Instead a copy of the text replaces the contents of the edit buffer.

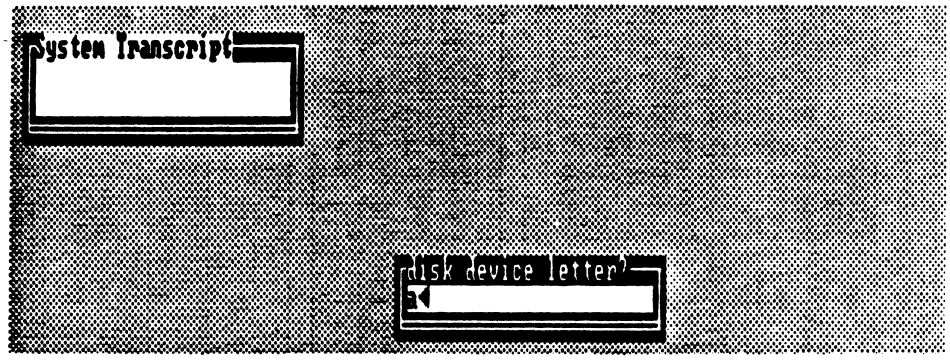
Text that is in the edit buffer can be pasted into the pane by either inserting it into a new place or replacing some existing text. To insert the contents of the edit buffer, place the insertion point at the desired position and then select the paste function from the pane menu. To replace some text with the contents of the edit buffer, first select the text to be replaced and then select the paste function from the pane menu.

The paste function leaves the contents of the edit buffer unchanged. This means that the same text can be pasted several times and in several different places.

Since the same edit buffer is shared by all of the text panes, it is a simple matter to transfer text between windows. First select the text to be transferred. Now place the text to be transferred into the edit buffer by using the pane menu to either cut it or copy it. Then move the cursor into the new window and

either place the insertion point where the text is to be inserted or select some existing text that is to be replaced. Now use the paste function from the pane menu. The text appears in the new window.

Prompters



Prompters are a special kind of text editing window that appear on the screen when you make a request that requires additional information. The label of a prompter is a message, the prompt, which tells you what information is desired. The single pane of a prompter is a text pane in which you can enter your response. There is often a default response in the text pane. The picture above shows the prompter that appears when you select the browse disk function from the system menu.

The text editor used by a prompter is the same as the text editor described under "Editing Text" above, except for the following differences.

Prompters are used by the environment to request a single piece of information from you. You must answer the request or tell the environment that you want to cancel whatever operation is requesting the information. Since you must respond to the request, Methods will not let you move the cursor outside of the text pane of the prompter. Also, there is no window menu for a prompter.

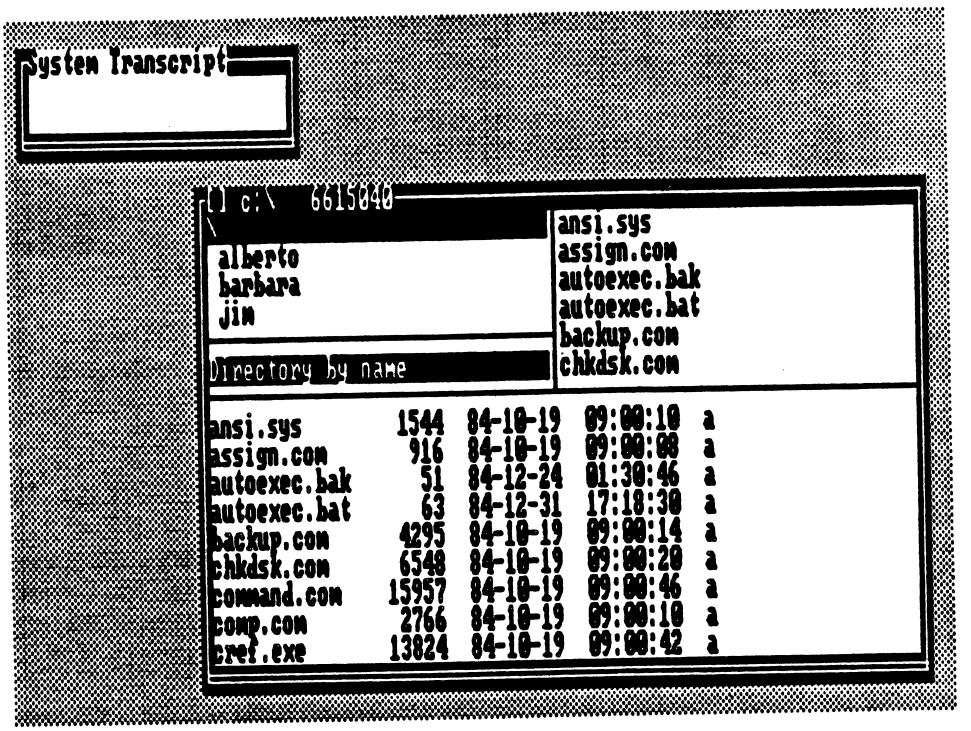
The pane menu of a prompter (which is popped up by pressing the pane menu key) has two choices which are specific to prompters. When you select the accept function, the prompter sends the text in the pane to the requestor of the information. When you select the cancel function, the prompter sends nil to the requestor. The requestor should cancel the operation. Pressing the return key is the same as selecting the accept function from the pane menu. After the request is answered, either by accepting or canceling, the prompter window disappears.

Browsing

List Panes

Many of the windows in the Methods environment have panes that allow you to select from a list of things. These are called list panes. Windows that have list panes are usually referred to as browsers.

Browsing the Disk



The Disk Browser shown above is an example of a browsing window that has two list panes as well as a text pane. This window allows you to see and modify all of the files on a disk device (in this example, those files on disk device C:).

To open a Disk Browser, pop up the system menu and select the browse disk function. Methods responds with a prompter that asks for the disk device character and suggests device A: as a default response. If you want device A:, just press the return key. If you want another device, press the backspace key, then the letter

of the device that you want, and then the return key. Methods will prompt you for a rectangle in which to open the window in the usual way (see "Defining a Rectangle on the Screen" on Page 15). After you define the rectangle in which you want the window to appear, Methods will display a Disk Browser for the disk device you specified.

Move the cursor into the list pane on the left and notice what happens. Methods reverses the line on which the cursor sits. This tells you that you are in a list pane. To select from a list, press the select key when the line that contains the item you want is reversed. What happens when you select from a list in a browser depends on the type of browser and the list pane that you select from.

Move the cursor to be on the line that has the backslash character (\) and select it. This selects the root directory on the disk. Methods responds by displaying the list of files in that directory in the list pane to the right and additional information about each file is placed in the text pane at the bottom of the window.

Move the cursor into the list of files (the list pane on the right) and select any file. You do this in the same way as you do for selecting the directory "\", since all list panes respond in the same way. Methods reads the contents of the file into the text pane at the bottom of the window. (If the file is not a text file, it may look strange). The Disk Browser is explained in greater detail in Chapter 3.

Most browsers work in this manner. When you select something from a list, the other panes in the window change to reflect the item that is selected.

Chapter 2: Programming in Smalltalk

Doing and Showing

This chapter explains how to execute Smalltalk expressions, how to write Smalltalk methods, and how the compiler and the Methods Environment report errors. The section on Disasters explains some simple steps that you can take to protect yourself from losing all your work in the event of a total failure (crash) of the Methods Environment.

Smalltalk is an object-oriented programming language. You program in Smalltalk by defining classes and methods, and then create objects and send messages to them. The details of the language and its syntax, and the concepts of class, method, object, and message are covered in the Smalltalk Programming Guide.

Briefly, the Methods Environment is a directed graph of objects of many different classes. A class defines the structure and behavior of all objects which are members of the class. A method specifies a message that is understood by the members (or objects) of a class. Messages are used in Smalltalk expressions. Messages are composed of a receiver object, zero or more argument objects, and a message selector which specifies the operation to perform. Every message produces an object as its result, similar to a function in other languages.

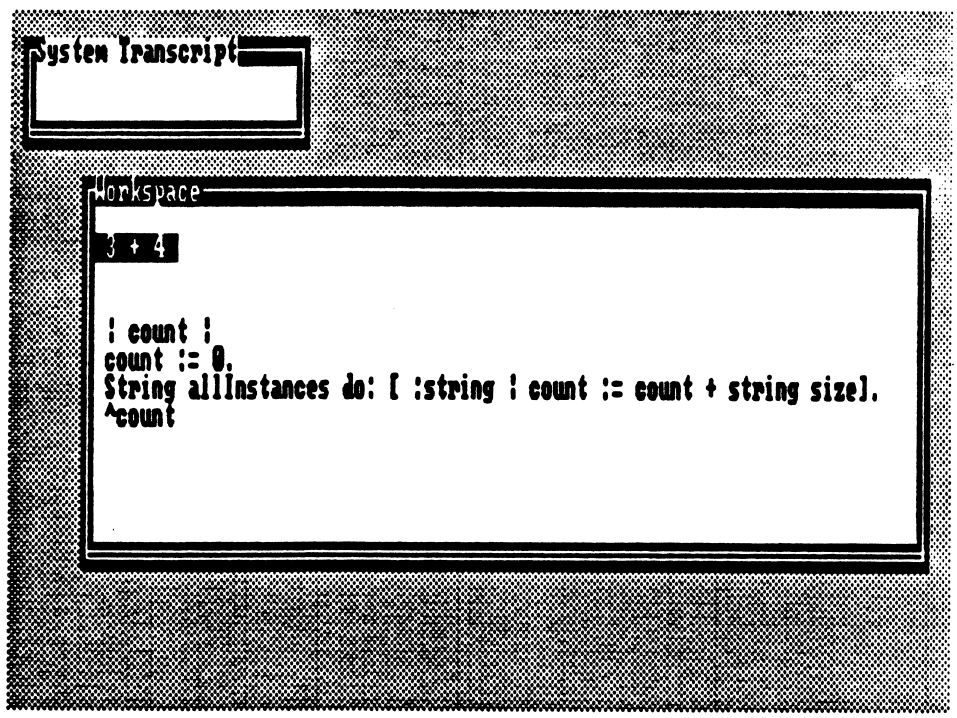
Some example messages and their result objects:

```
1 + 2                   receiver:  SmallInteger 1
                        arguments: SmallInteger 2
                        selector:  +
                        result:    SmallInteger 3

'hello' at: 5           receiver:  String hello
                        arguments: SmallInteger 5
                        selector:  at:
                        result:    Character o

12 factorial            receiver:  SmallInteger 12
                        arguments: none
                        selector:  factorial
                        result:    LargePositiveInteger
                                        838252800
```

Note: Methods supports variable length integers with up to approximately 500000 (yes, 1/2 million) bits of precision. Of course, the bigger they are, the slower the computations.

```
5 between: 3 and: 7        receiver:   SmallInteger 5
                           arguments:  SmallInteger 3
                                       SmallInteger 7
                           selector:   between:and:
                           result:     true
```

Later in this chapter in the section on the Class Browser, and also in Chapter 4 in the section on the Class Hierarchy Browser, we explain how you can find and look at the Smalltalk code used in the above expressions (and any other expression you evaluate).



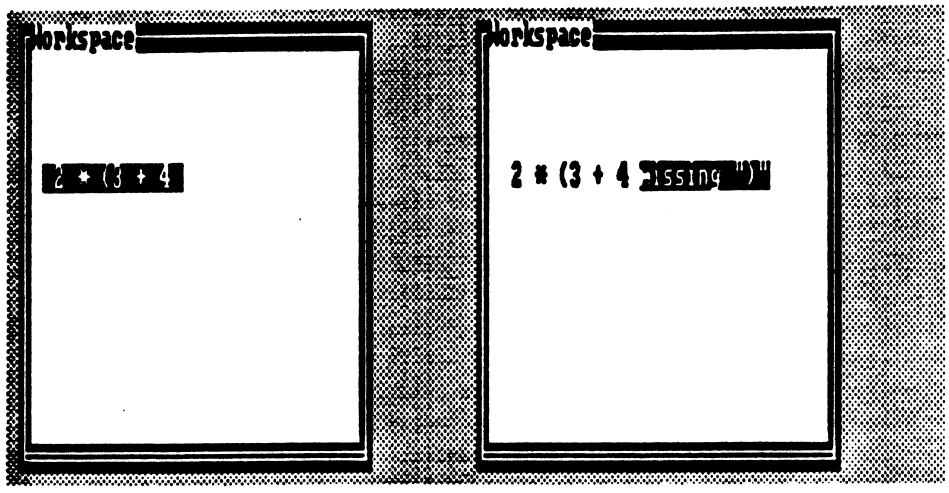
While the foregoing description seems formidable, in practice it is very simple and easy to use the language. All text panes in the system support immediate expression evaluation via their pane menus. The picture above shows a Workspace with the expression 3+4 selected. To evaluate an expression, you must first select

it, then pop up the pane menu and select either the do it or the show it function. If show it is selected, the expression is evaluated and a character representation of the expression value is inserted in the pane after the evaluated text. If do it is selected, the expression is evaluated and the expression value is thrown away. Notice that only the text that is selected is evaluated, the other text in the pane is ignored. Extra blanks at the beginning or the end of the selected text are similarly ignored.

Any legal Smalltalk expression or expression series can be selected in a text pane and evaluated. Temporary variables can be declared as needed. Also shown is a series of expressions that are not selected. The first line of the text declares a temporary variable. The second line initializes it to zero. The third line iterates across all the strings in the environment and accumulates the string sizes in the temporary variable count. The fourth line returns count as the expression series result. To evaluate this series of expressions, select all four lines of text and then use the show it function on the pane menu.

You may notice the disk being accessed from time to time when you evaluate expressions. Methods is logging all of the expressions that are evaluated onto the change log file (change.log). This feature makes it much easier to recover most of your work in the event of a system crash or other disaster. The section on Disasters explains more about this capability.

Compilation Errors



When you select and evaluate an expression, Methods compiles it and then executes it. The picture above shows what happens when there is a compilation error. The window on the left shows an expression before it is evaluated, and the window on the right shows what the same Workspace would look like after attempting to evaluate this expression.

The selected text in the screen on the right is the error message generated by the compiler and it is inserted by Methods at the point of the error. Merely edit the text to fix the error (in this case, just type the ")" character and it will replace the error message) and then select and evaluate the text again.

Making Command Templates

It is not necessary to type the expression that you want to evaluate. You can edit some existing text in the pane and then select and execute it. This feature makes it possible to build a window or even a file of useful expressions that you can edit as command templates before executing.

The Image disk contains a file, sample.sml, with several useful expressions in it. You can use the Disk Browser to look at and also execute the expressions contained in this file. These expressions all have comments explaining them.

You can add to the expressions in this file (or make files of expressions of your own) using the Disk Browser. You can include comments with your expressions by merely enclosing the comment in double quotes: "this is a comment".

Using Global Variables

Methods has a dictionary containing all of the global variables in the environment. This dictionary is named Smalltalk and is itself a global variable. To define a new global variable named Friend, evalutate the following expression (use the do it function on the pane menu):

```
Smalltalk at: #Friend put: nil
```

Global variables must begin with an uppercase letter. The '#' character is the Smalltalk way of defining a symbol.

If you use the show it function when you create a new global variable, you will notice that a lot of text is inserted into the pane. This is because every Smalltalk expression returns a result (like a function in other languages). Evaluating using show it causes Methods to print the expression result. The result of the above expression is the the global dictionary "Smalltalk", so the environment prints the entire dictionary. Using do it for these kinds of expressions avoids this. (If you do use show it, no harm is done; you can delete the unwanted text).

You can print the current value of any global variable by selecting the text of the variable name and then using the show it function. For example, if you define the global variable Friend, you can print its current value by selecting the following text and then using the show it function:

```
Friend
```

You can find out the class of any object, including the current value of a global variable by evaluating an expression like the following example with the show it function:

```
Friend class
```

You can change the value of a global variable by assigning a new value to it. For example, the following expression will assign a new instance of the class Example to the global variable Friend:

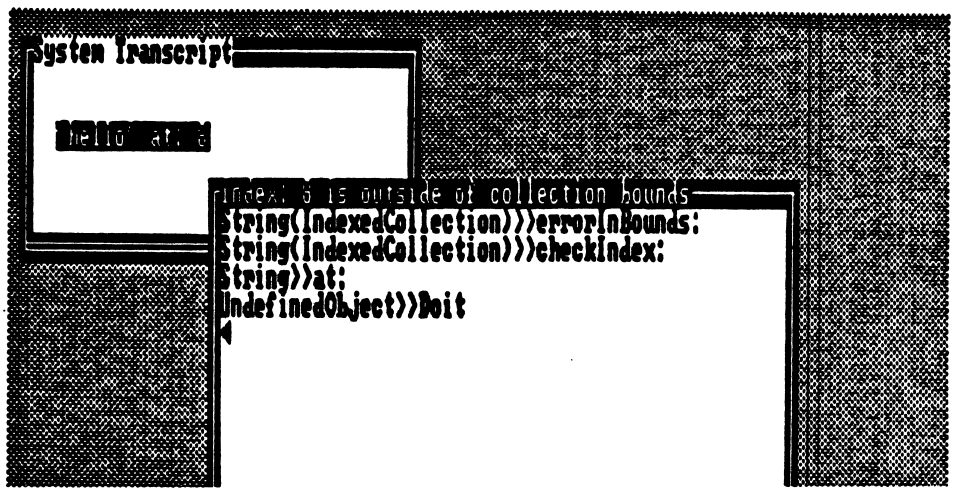
```
Friend := Example new
```

Alternatively, you can alter the value of a global variable or define a new global variable with an initial value by evaluating an expression like:

```
Smalltalk at: #Friend put: Example new
```

You may have noticed that the class Example is used in expressions the same way as global variables are used. This is because classes are contained in global variables.

Execution Errors - Walkback Window



The picture above shows what happens if an error is detected when an expression is executed. In this example, the expression executed is the selected text in the System Transcript. This expression attempts to access the eighth character in the string 'hello', obviously in error. The window on the right is the Walkback window that notifies you of the error when it is encountered during the evaluation of the expression.

A Walkback window is placed on the screen by Methods whenever an execution error is detected. The label of the window has a message describing the error. The text pane of the window contains a method walkback showing the sequence of incomplete message sends (function calls) that led to the error.

Each line in the text pane represents a single message send. The class name of the receiver is given first. If the method was inherited from a superclass, the class name where the method came from is displayed next, enclosed in parenthesis. The string on the right after the '>>' is the message selector (name of the method).

The Smalltalk Language Guide gives detailed descriptions of message send, receiver, class, superclass, and inheritance. In simple terms, a messge send is like a function or subroutine call. The walkback is a trace of all incomplete (nested) message sends that led up to the error. The innermost message send is listed first. The message selector identifies the name of the

method that is being executed, similar to a function or subroutine name. The combination of the selector and the class name of the receiver specifies the particular method implementation that the message send is using. Since objects can inherit methods from their superclasses, the implementation of the method may be from one of the object's superclasses. In this case, the class name of the superclass in which the method is actually implemented is given.

As is explained in the section on the Class Browser in this chapter and the Class Hierarchy Browser in Chapter 4, you can look at the source code of the methods shown in the Walkback. Using either of these two browsers you can also modify the source code to remove the problem and recompile the method.

The Walkback window stays on the screen until you close it, just like all other windows. You do not have to close it before you do something else; just move the cursor into another window.

Disasters

Methods is very resiliant to errors. However, disasters can and will happen, especially if you are making modifications to Methods. This section attempts to explain what kinds of things can go wrong and some simple steps for recovery. These same issues are discussed in detail in the section on Surviving a System Crash in Chapter 5.

Methods uses a stack machine architecture in the Smalltalk interpreter. If you overflow the stack, a Walkback window will appear. The list of incomplete message sends should identify the method that is recursing improperly. You can correct the method using either the Class Browser described in the following section or the Class Hierarchy Browser described in Chapter 4.

If you send a message to an object that does not implement or inherit a method to perform the message, Methods will produce a Walkback window. The label of the Walkback will tell you the message is not understood. This is the most common kind of runtime error. By examining the list of incomplete message sends in the Walkback window, you should be able to determine your problem. Usually it is one of two things: (1) you have spelled the messages selector incorrectly or (2) the receiver object is

not what you think it should have been. Correct the spelling or recompile the bad method with either the Class Hierarchy Browser or the Class Browser.

If Methods does not respond when you type or move the cursor, it could be looping. If you evaluate an expression like the following one, Methods loops indefinitely:

```
[true] whileTrue: []
```

You can break the loop by typing a cntrl-break. This causes a user interrupt and will display a Walkback window. You should be able to identify the looping method from the list of message sends.

These errors are fairly easy to recover from since Methods is still running. There is a more serious error from which Methods cannot recover. If Methods cannot find the code that displays the does not understand Walkback window, something very serious has gone wrong and Methods terminates with the error message: recursive does not understand. Methods has crashed. If you have taken the precautions described below, then by reading the section on Surviving a System Crash in Chapter 5 you should be able to recover most of your work.

As explained earlier, Methods logs all expressions that are evaluated and all changes to the classes and methods in the system onto the change log file (change.log). In addition, the image file represents a working version of the system. It is the version of the system that Methods will use when you start the environment. The source file (source.sml) has all of the source code of the base system. You should maintain a backup version of these three files that is known to work. This will provide a fall-back position in case of any serious disaster.

In addition, you should save the image onto the disk (you can do this from the system menu) before trying something that might crash the system. This will preserve most of your work. If the system crashes, you can restart Methods using this saved image. If you fail to save the image and the system crashes, you can use the Disk Browser to examine the change log. As explained in "Recovering from a System Crash" in Chapter 5, you can install most of your unsaved work back into the system quite easily.

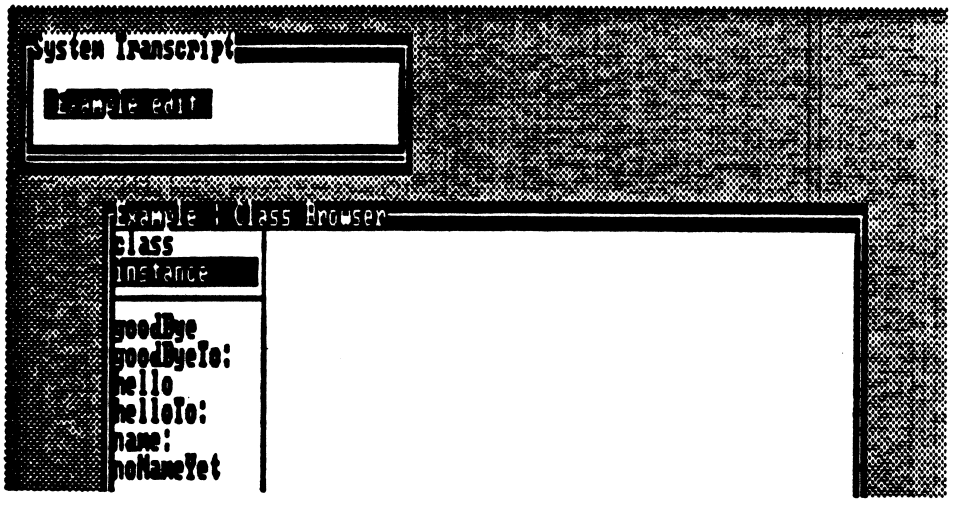
Class Browser

Opening a Class Browser

A Class Browser lets you browse, modify, and add to the methods for a particular class. You can open a Class Browser in either of two ways. You can send the message edit to any class or you can select the browse function from the hierarchy pane menu of the Class Hierarchy Browser described in Chapter 4.

Move the cursor into any text pane (the System Transcript, for example), type in the text Example edit and select it (make it reversed). Example is the name of a class in Methods (the class used for the examples in this chapter). Select the do it function from the pane menu. Methods responds with a blinking rectangle. After you define the location and size of the new window, Methods opens a Class Browser for the class Example.

Appearance



The picture above shows a Class Browser for the class Example. Class Browsers have three panes: the dictionary list pane, the method list pane, and the contents pane.

The label of a Class Browser identifies the class that is being browsed. Only the methods for this class can be browsed, added, or modified in the window.

The dictionary list pane is the top left pane that contains the two choices: class and instance. This pane lets you choose whether you wish to browse class or instance messages.

The methods list pane is immediately below the dictionary list pane. It displays the list of class or instance methods implemented by the class. Which list is displayed is determined by which choice (class or instance) is made in the dictionary list pane. This pane lets you choose a particular method implementation for viewing.

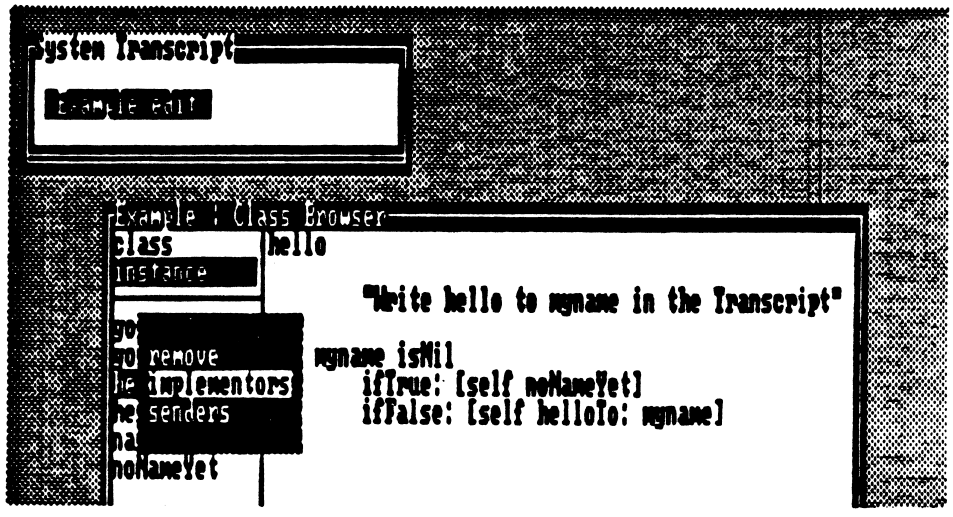
The contents pane is the large pane to the right of the two list panes. It displays the currently selected method in the methods list pane. You can edit and recompile methods in this text pane.

Instance and Class Methods

The dictionary list pane contains two choices, class and instance. If you select instance (by placing the cursor on top of instance and then pressing the select key), then the list of instance methods will be displayed in the method list pane immediately below this pane. If you select class, then the list of class methods will be displayed in the method list pane.

Instance methods define messages which are understood by instances of the class. Class methods define messages understood by the class as a whole. In the class Example, the message hello asks an instance of Example to print a greeting on the System Transcript. The class message named: asks the class Example to create a new instance of itself with a particular name.

There is no pane menu associated with the dictionary list pane. If you press the pane menu key, Methods will pop up the same menu that the window menu key pops up.



The picture above shows the pane menu for the methods list pane, the pane on the lower left of the Class Browser window. The functions on this menu let you remove methods and display the implementors (definers) and the senders (callers) of messages.

The methods list pane displays a list of the message selectors for either the class methods or the instance methods defined in the class. The choice you make in the dictionary list pane (immediately above this pane) determines which list you see. Notice that the choice you make is highlighted in the dictionary list pane. If you want to view the other list, just move the cursor over the other choice, class or instance, and press the select key.

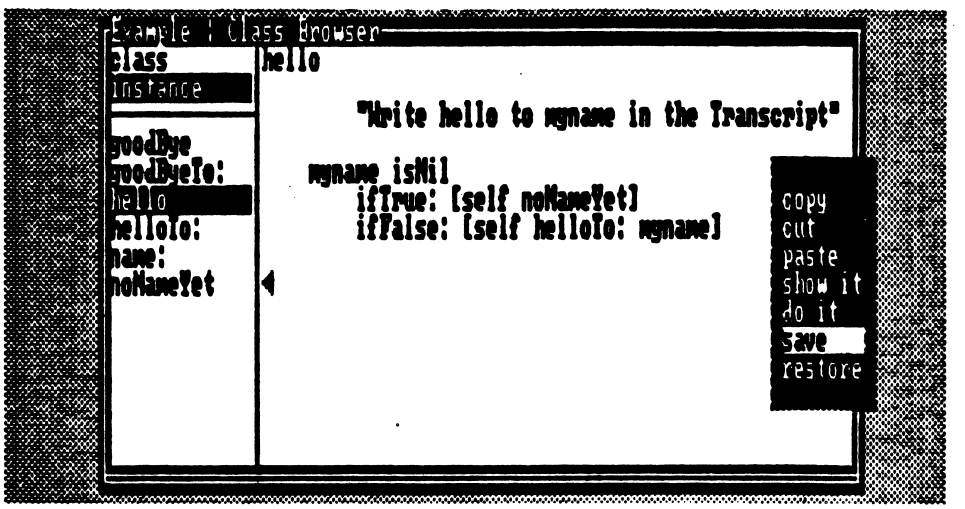
To select a method, move the cursor into the methods list pane and press the select key when the cursor is on top of the message selector you wish to select. Methods responds by placing the source code for the method that implements the message in the contents pane. Notice that as you move the cursor in a list pane, the item that is under the cursor is reversed. This is the method that will be selected when you press the select key. The method list pane, like all list panes, can be scrolled vertically with the vertical scrolling keys, Pg Up and Pg Dn.

To remove a method you must first select it. The remove function on the pane menu removes the currently selected method. If no method is selected, nothing happens.

Senders causes Methods to search all of the methods in the environment for senders (callers) of the selected message selector. A Senders Window pops up to display the classes and methods that use the selected message selector.

Implementors causes Methods to search all of the classes in the environment for implementors (definers) of the selected message selector. An Implementors Window pops up to display the classes that implement methods for the selected message selector. The Number class and its subclasses contain many examples of mulitple implementations for a message. A simple example is the + message. Adding Fractions is very different from adding Integers.

Modifying and Adding Methods



The picture above shows the pane menu for the contents pane, the large pane on the right side of the Class Browser. The contents pane is a text pane used for modifying and adding methods to the class being browsed.

The contents pane displays the source code of the currently selected message selector in the methods list pane. To see the source code of a different method, move the cursor into the methods list pane and select another message selector.

The functions cut, copy, and paste on the pane menu behave exactly as described under "Cutting, Copying, and Pasting" in Chapter 1.

The functions do it and show it on the pane menu behave exactly as described earlier in this chapter under "Doing and Showing". To modify a method, first select the message selector in the list pane. Then edit the source code in this pane (the contents pane). To add a new method, replace the text in the contents pane with the source code of the new method. In either case, you invoke the compiler with the save function on the contents pane menu. If there is a compilation error, the error message will be inserted into the source code at the place where it is detected. Merely edit the source code and use the save function again.

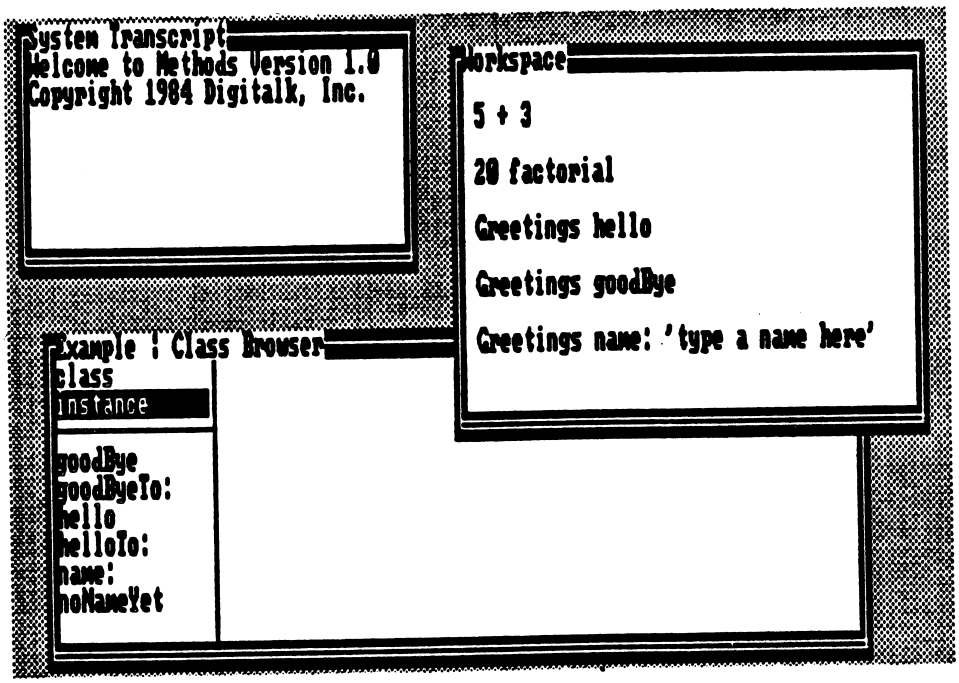
When the method is successfully compiled with no errors, Methods installs it into the class automatically and all future invocations of this method use this new version. The source code of the new or modified method is written onto the change log file (change.log). The logging of the changes you make to the system make it much easier to recover from an unexpected system crash. For a fuller explanation, see "Disasters" in this chapter.

The restore function on the pane menu causes Methods to retrieve the current source code being used by the system for the selected method (latest version that has been successfully compiled). The source code is automatically read from either the source file (sources.sml) or the change log file (change.log).

Saving Your Work

As explained in the section "Exiting and Restarting" in Chapter 1, none of the changes you make to the system are made permanent until you save the image. If you have modified or added any methods to the environment, you must save the image to make them permanent. You can save the image by selecting save image from the system menu or by saving the image as you are exiting the environment.

Examples



The picture above shows a Workspace with some sample messages in it. Open a Class Browser on the class Example and then evaluate these messages one at a time in some Workspace. Use the Class Browser to see the code that is used in evaluating these messages.

In particular, make a global variable Friend by evaluating (using do it) the expression:

```
Smalltalk at: #Friend put: nil.
```

Then make an instance of class Example with your name by evaluating:

```
Friend := Example named: 'your name goes here'.
```

You can use this instance of class Example to try out the methods defined in this class.

If you want to see some error handling examples, evaluate either or both of the following messages:

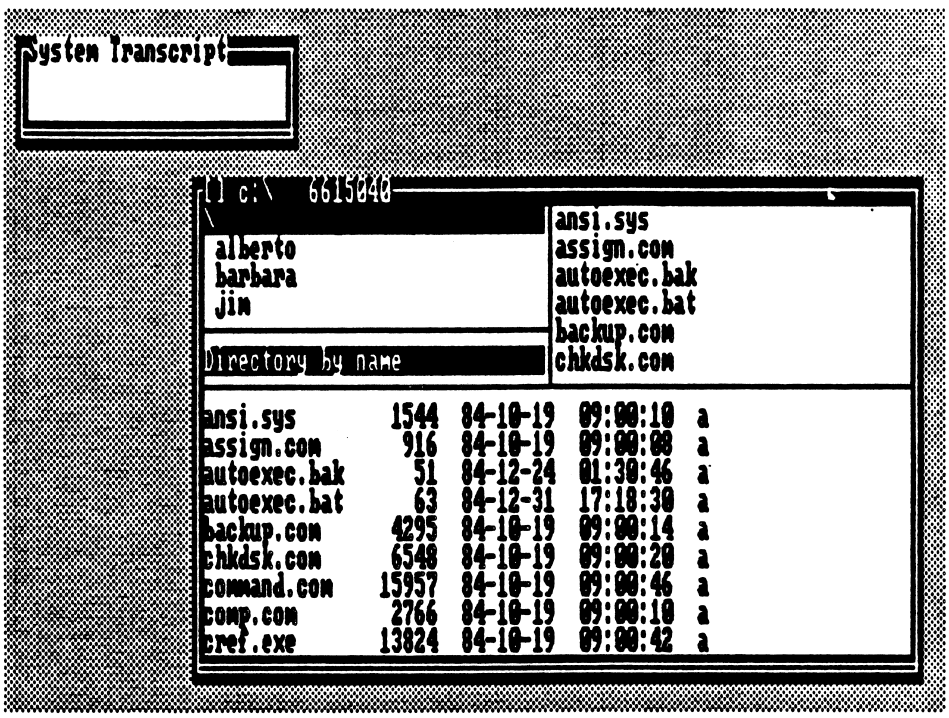
```
Friend name: 2.

Friend := Example named: 2.
```

Notice that the errors are reported in different ways. Look at the methods in class Example to see how this is accomplished. Remember that name: is an instance message, and named: is a class message.

Chapter 3: Browsing Disks

Opening a Disk Browser

The Disk Browser allows you to browse all of the files on a disk device. To open a Disk Browser, select the browse disk function on the system menu. Methods will respond with a prompter asking you for the disk device character. After replying to the prompter and defining a rectangle for the new window, Methods will open a new Disk Browser. You can have as many Disk Browsers open at a time as you want.



The Disk Browser is divided into four panes as shown above.

The directory hierarchy list appears in the upper left pane. In this pane the names of all the directories on the disk will be listed. You will see a backwards slash symbol in the top corner of the pane. This symbol stands for the root or parent dictionary of the entire disk. The names of the directories in

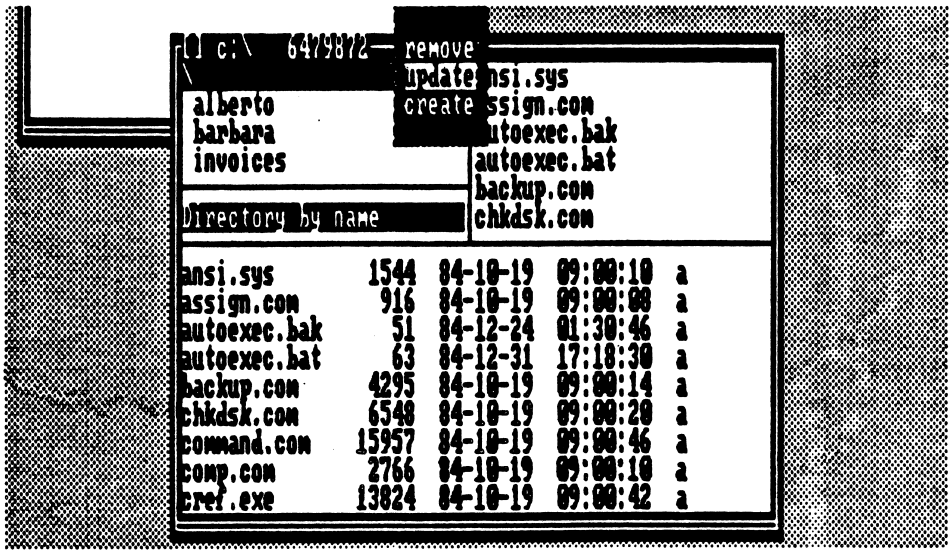
the parent dictionary are indented and appear in hierarchal order. The names of any sub-directories will be further indented.

The file list is in the pane to the right of the directory hierarchy list. When a directory is selected in the directory hierarchy list pane its files will appear in this pane.

The contents pane exhibits the text of a selected file or further directory information if no file has been selected.

The directory order pane contains a statement telling you how the information in a directory is ordered when it is displayed in the contents pane. The pane menu associated with the directory order pane gives you the choice of ordering the directory by date, name, or size.

The label of the Disk Browser displays other useful information. If the disk being browsed is labeled (disk labeling is described in your PC or MS DOS manual), the label is displayed between the brackets "[" and "]". The full pathname of the currently selected directory in the browser is displayed. If no directory is selected, then just the device drive characeter appears. The numbers at the right side of the label are the amount of free space on the disk. This is automatically updated as you use the browser.



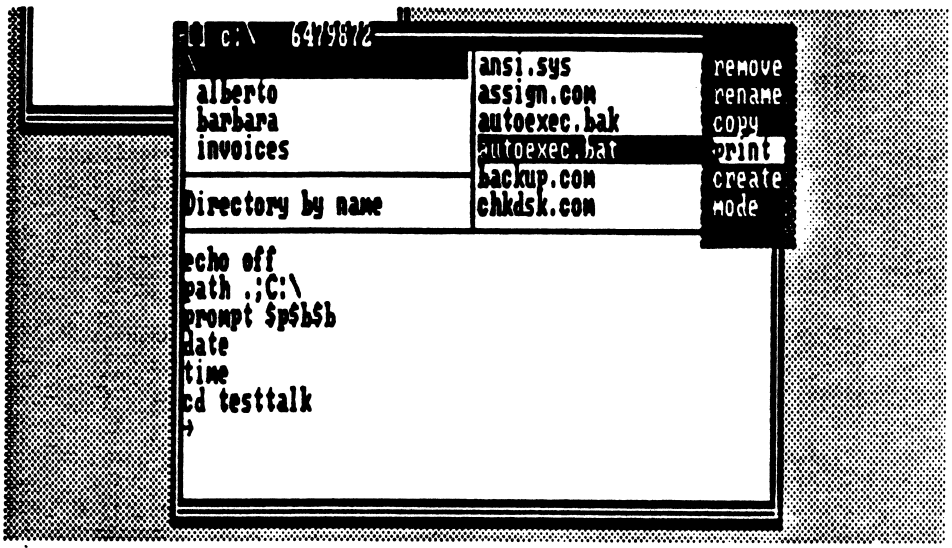
The picture above shows the menu for the directory hierarchy list pane, the pane at the top left of the disk browser window. The functions on it permit you to either make modifications or bring up another disk's directories.

To select a directory to browse, move the cursor into the directory list pane and press the select key when the cursor is on top of the directory you wish to browse. Methods will display the list of files in the file list pane (the right hand pane) and a complete listing of the directory will appear in the contents pane (the bottom pane). You can move the cursor and/or scroll the directory list pane and select a different directory any time you wish.

Remove causes a selected directory and its files to be eliminated from the disk.

Update allows you to browse another disk in the same drive. Selecting this function tells the window to update itself by rereading the directory structure on the disk. This function should be selected after the second disk has been inserted.

Create causes a new directory to be made. After this function is selected a prompter will come up which asks for the path (directory) name. Type in your response and press the return key to make it appear in the pane.



The menu for the file list pane shown above displays a variety of functions for files. The file list pane is the top right pane of the disk browser window.

To select a file to browse, move the cursor on top of the file name in the file list pane and press the select key. Methods responds by displaying the file contents in the contents pane. You can select another file to browse anytime you wish by selecting it in the file list pane.

Remove causes a selected file and its contents to be eliminated from a disk.

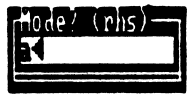
Print causes the file to be printed.

Mode permits you to change the system file mode bits (see your PC/MS-DOS manual).

Rename permits you to alter the name of an existing file.

Copy causes a selected file to be duplicated in another location.

Create causes a new file to be generated.



Pictured above are the three prompters for the file list pane. If you select the rename or copy function of the pane menu, you will see the prompter shown at left.

The create function causes the prompter at center to appear. The "filename.ext" in the response area serves as a reminder that the file name is limited to 11 characters and that a period must be inserted between the file name and extension. Edit the pane contents until it contains the file name you want.

The prompter at above right pops up when the mode function is selected. Again, you are referred to your PC/MS-DOS manual.

Above is a picture of the three menus for the file contents pane, the pane at the bottom of the window. Which menu appears is dependent upon what is selected in the other panes.

If you are looking at a directory, you can pop up the menu shown at center. The menu at left can be brought up if only part of a large file has been read into the pane. You can pop up the menu on the right if you have selected a smaller file. All of these menus are popped up by pressing the pane menu key (Methods knows which one to use automatically).

As you can see, the three menus share some common functions. The copy, cut, and paste functions are the same as those used in normal text editing (described in Chapter 1). In addition, the do it and show it functions are identical to those previously described for Smalltalk expression evaluation. Save and restore have been explained under text editing as well.

The other functions of the file contents pane menus are the following:

Read it is used to read the entire contents of a large file. Normally, when the text of a file exceeds 6000 bytes, the pane will only display the first 2000 bytes, followed by the final 2000 bytes. After you select this function, the entire file will be read into the pane.

Save as is used to save edited material in a file different from the selected one, as opposed to save which rewrites the selected file with the text in the pane. Selection of this function causes a prompter to appear which asks for the file name. You must respond with the name of a new or existing file.

Install compiles the selected text into the system. The selected text must be in the same format as the text in the change log and text in a class definition file. It is used to execute Smalltalk expressions and to compile and install Smalltalk source code from files. Typically, these would be from the change log or from a file containing a class definition. This use of the change log is described in more detail in Chapter 5. Class definitions files can be made using the Class Hierarchy Browser (see the file out function under "Browsing Classes" in Chapter 4).

The directory order pane, located immediately below the directory list pane, allows you to see the full information associated with each file in a directory, as well as control the order in which this information is displayed.

To display the directory in the contents pane, move the cursor into the directory order pane and press the select key. Methods responds by displaying in the contents pane information about the files in the currently selected directory (the one selected in the directory list pane). To display the files in the directory in a different order, pop up the pane menu of the directory order pane by pressing the pane menu key. You now have a choice of ordering by date, name, or size. After you make a choice, Methods will reorder the files.

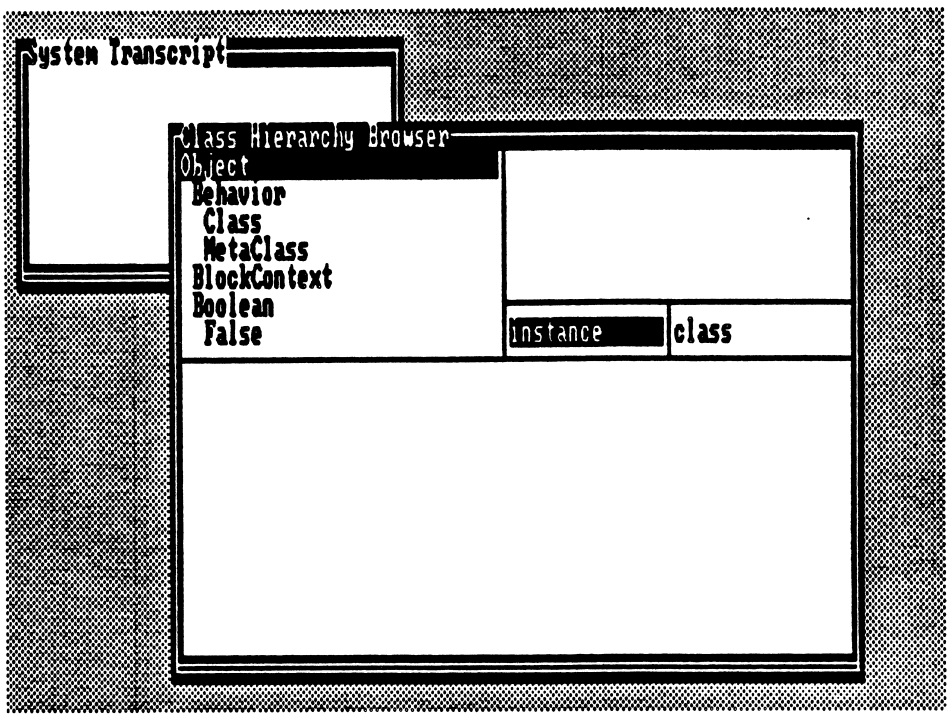
The **save as** function in the pane menu of the contents pane allows the directory information to be written on a file. You can also edit the information in the contents pane (for example, adding file comments) before writing it on the file.

Chapter 4: Modifying Methods

Opening a Class Hierarchy Browser

Select the browse classes function of the system menu to open a class hierarchy browser. A blinking rectangle appears. Define the location and the size for the new class hierarchy browser window (as described under "Defining a Rectangle" in Chapter 1). A large window is recommended.

The class hierarchy browser gives you the ability to look at, and, if desired, to modify Smalltalk code and the classes in the system. As opposed to the class browser, described in Chapter 2, in which only one class is visible, the class hierarchy browser gives accessibility to the code for all of the classes. More important, the relationships between the classes can be seen, giving a big picture of the system. These relationships are in the form of a hierarchy of classes, subclasses, subsubclasses, and so forth.



The class hierarchy browser is divided into five panes.

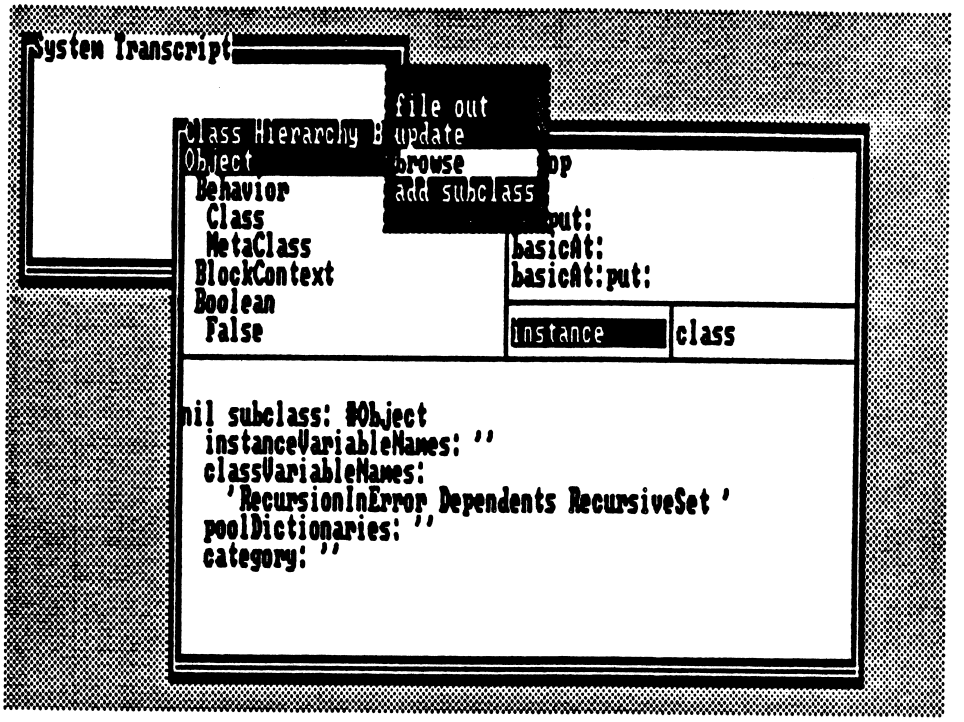
The class hierarchy list appears in the upper left pane. In this pane, the names of all of the classes in the system are presented in a hierarchical order. Notice that Object appears first in the list. Class Object is outermost in the hierarchy. All other classes are subclasses to Object and therefore appear indented. Subclasses of these classes are indented further.

The method list pane is the pane to the right of the class hierarchy list. There are two kinds of methods which can be listed, instance methods and class methods. Instance methods define messages which are understood by instances of the selected class. Class methods define messages understood by the selected class as a whole. Using class Time as an example, a method that subtracts two times would be an instance method, and a method that computes the time it takes to execute a block of code would be a class method.

The two little panes underneath the method list pane are used to select either the list of class methods or the list of instance methods in the method list pane. The choice is made by moving the cursor over either the class pane or the instance pane and then pressing the select key.

The Smalltalk code itself is seen in the contents pane which occupies the bottom half of the screen. When you select a class in the hierarchy pane, the message which defines the class selected is displayed in this pane. Select a method and its code appears.

Browsing the Hierarchy



The picture above shows the menu for the class hierarchy list pane, the top left pane of the window. The functions on it permit you to write the definition of a class and all of its methods on a file, browse a particular class, and add a subclass.

File out writes the class definition along with all of the instance and class messages of the selected class to a file. Methods derives the file name from the class name with the extension .cls and places it in the directory in which Methods was invoked. Subclasses are not automatically filed out.

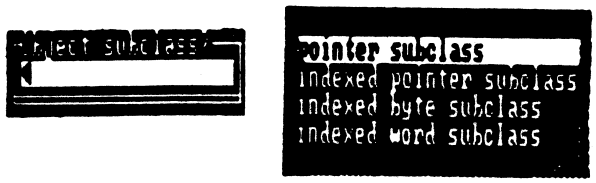
To select a class to browse, move the cursor into the class hierarchy list pane and select a class by pressing the select key. Methods displays the class definition in the contents pane and a list of methods implemented by the class in the methods list pane. You can move the cursor and/or scroll the class hierarchy pane and then select a different class by pressing the select key.

Update tells the class hierarchy browser to recompute the class hierarchy list. This becomes necessary if you have added or removed classes not using this browser.

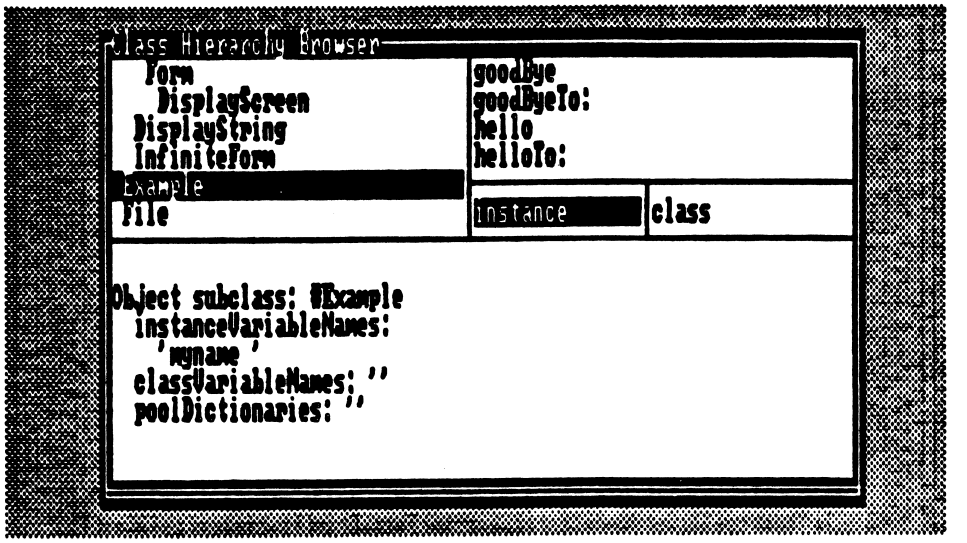
Browse creates a class browser (see Class Browser in Chapter 2) for the selected class.

Add subclass permits you to add a subclass to the selected class.

Adding Classes



To add a subclass of a class, first select in the hierarchy list pane the class that will be the superclass of the new class. Now pop up the pane menu and select the add subclass function. The prompter shown above appears asking for the subclass name. The superclass name (the one currently selected) appears in the prompter label. When you reply to the prompter, Methods responds with the menu shown above asking you the type of subclass that you want, see the Smalltalk Language Guide for a description of class types. After you make a choice, the new subclass is built and the class hierarchy list is automatically updated. This subclass may now be selected in order to define its instance and class variables, add methods to it, and/or define subclasses of it.



To define the instance variables, class variables, and pool dictionaries of a class, you must first select the class in the class hierarchy list pane. Methods responds by displaying the current class definition in the contents pane. The Smalltalk Language Guide gives more detailed information about class definitions. The picture above shows the class definition for class Example. You change the class definition by editing the text in the contents pane and then selecting the save function from the contents pane menu.

When you change a class definition, Methods automatically recompiles all of the methods in the class and all of its subclasses. In addition, a message is written on the system change log giving the new class definition.

Changes to a class definition take effect immediately. This means that all future instances of the class will have the new structure. Therefore, special care must be taken in modifying classes that are used by the Methods environment itself. You are actually changing the environment. For this reason, it is advised that subclasses be defined in preference to modifying the structure of existing classes until you are confident of what the effects will be.

Removing Classes

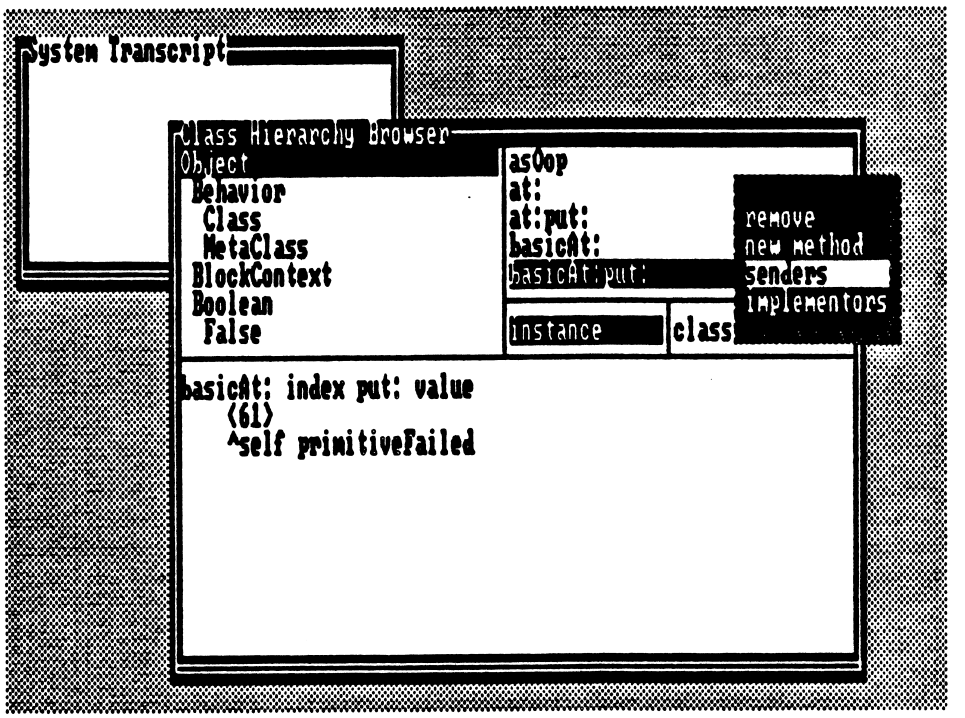
There is no remove function on the pane menu. To remove a class you must evaluate the following expression in any text pane (the contents pane of the Class Hierarchy Browser or the System Transcript, for example):

```
NameOfClass removeFromSystem
```

All of the methods in the class are automatically removed when the class is removed.

Methods will not let you remove a class if there are subclasses or if there are instances of the class anywhere in the environment. If either of these exist, you will get a Walkback window explaining the problem. Remember that Smalltalk automatically will remove (collect as garbage) any object that is not referenced by some other object in the system. Therefore, to remove all instances of a class, you must change all references to instances of the classes to refer to something else.

Browsing the Methods



The picture above shows the menu for the method list pane, the top right pane of the window, which contains functions to manipulate the methods of a class. Remember that there are two kinds of methods in a class, instance methods and class methods. The list that appears is determined by the two panes directly beneath the method list pane, the class and the instance panes. You select the list by moving the cursor over the class or instance pane and then pressing the select key.

To select a method, move the cursor on top of the method name (or more precisely, the method selector) in the method list pane and then press the select key. Methods responds by displaying the source code for the method in the contents pane below. Methods may have to read the disk in order to retrieve the source code, since all of the source is contained in the source file (sources.sml) or the change log file (change.log).

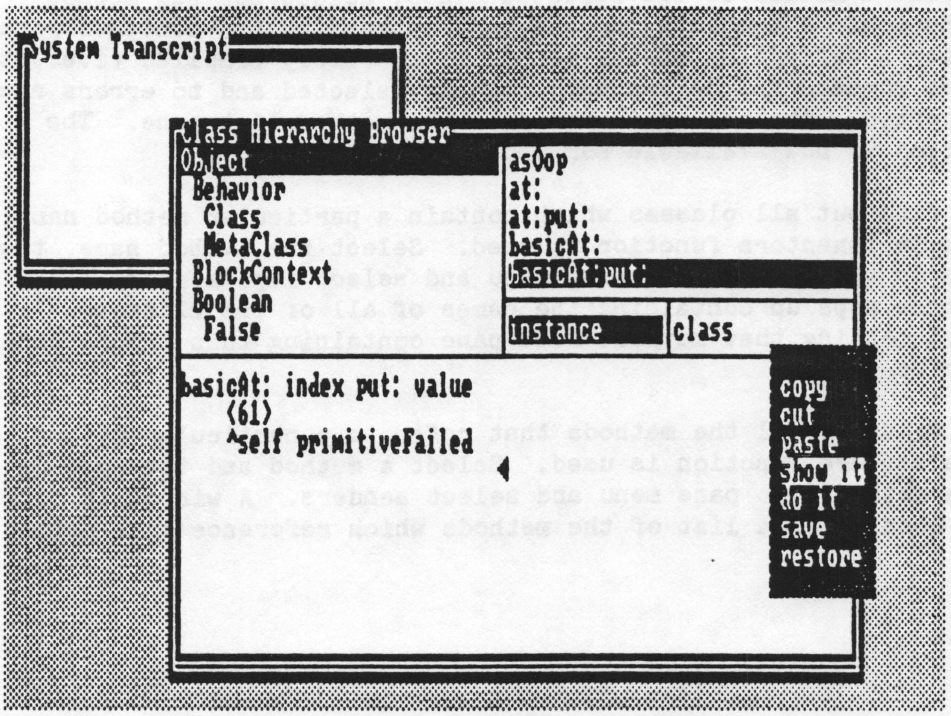
To remove a method from the selected class, select the method; pop up the method list menu; and select the remove function. The method list is immediately updated.

To add a method to the selected class, select the new method function. A template for a method appears in the contents pane. After a definition of the method is correctly compiled (i.e., the save command in the contents pane is selected and no errors are reported), the method is added to the method list pane. The method is now available for use.

To find out all classes which contain a particular method name, the implementors function is used. Select the method name, then pop up the method list pane menu and select implementors. A window pops up containing the names of all of the classes implementing that method. The pane containing this information may be edited.

To find out all the methods that refer to a particular method, the senders function is used. Select a method and then pop up the method list pane menu and select senders. A window is popped up containing a list of the methods which reference this method.

Adding and Modifying Method Source Code



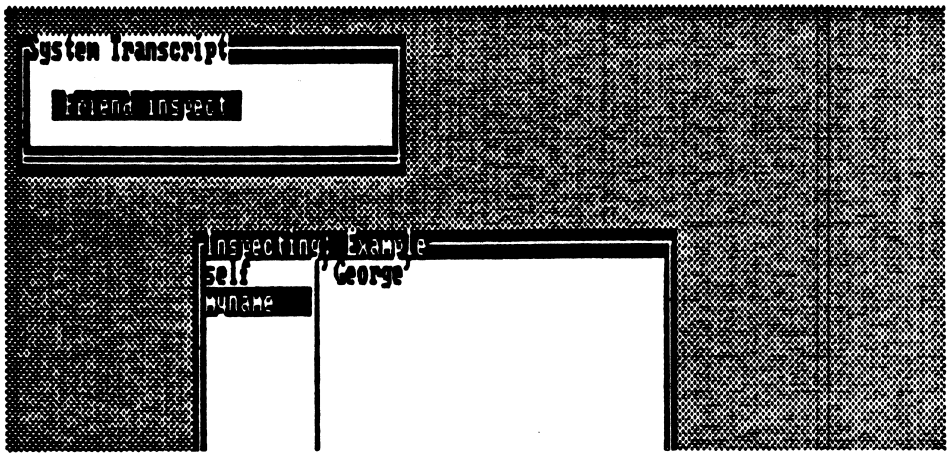
The picture above shows the pane menu for the contents pane. All of the functions on this menu behave exactly as described under "Editing Text" in Chapter 1 and "Doing and Showing" in Chapter 2.

To modify an existing method, you must first select it in the method list pane. The source currently being used by the system is displayed in the contents pane. After editing the source text, you then use the save function on the contents pane menu to invoke the compiler and install the new method in the environment.

To add a new method, you can either use the add method function in the pane menu of the method list pane to get a template that you can then edit, or edit the text of any existing method in the class. In either case, you use the save function to invoke the compiler and install the new method. Remember that there are two kinds of methods, class and instance methods. The kind of method you get is determined by which of the two little panes, class and instance, directly above the contents pane is currently selected.

If a compilation error is detected, a message is inserted into your source in the pane at the point where it was detected. The error message is selected (reversed). Just edit the text and use the save function again to recompile.

After a successful compilation, Methods writes the text of the new method on the change log file. A pointer in the compiled form of the method points to the file location where the source is stored. For this reason, your changes are not permanent until you save the image (see "Automatic Logging of Changes" and "Importance of Saving the Image" in Chapter 5 for more details). To save the image, either use the system menu function save image or save the image while exiting.



The picture above shows an Inspector window. Inspectors are used to examine and change objects in the system. They are a low level debugging aid.

To open an Inspector, send the message inspect to any object. For example, to open an inspector on the global variable Friend evaluate the following text:

```
Friend inspect.
```

Inspectors have two panes. The list pane on the left is the instance variable list pane. The pane on the right is the instance variable value pane.

The instance variable list pane shows all of the instance variables of the object being inspected. When you select one of the variables from the list, its current value is displayed in the instance variable value pane. The pane menu has only one function, inspect. When you select this function, a new Inspector will be opened on the currently selected instance variable.

The instance variable value pane is a text pane. You can use this pane to evaluate any type of expression that you wish. There are two very interesting features of this pane:

1. Any expression that you evaluate is compiled in the scope of the object being inspected. What this means is that you can use the current values of all of the instance variables in your expressions.

2. If you select the save function from the pane menu, the entire contents of the text pane will be compiled and executed. The result of this expression will replace the current value of the selected instance variable in the list pane to the left. Similarly, if you select the restore function Methods places the current value of the selected instance variable into this text pane.

Chapter 5: Maintaining Methods

Automatic Logging of Changes

As you define and modify classes or methods, Methods is logging all of these changes to the change log (the file change.log). Methods maintains pointers from the compiled methods in the environment to the source code in the change log. That is why the disk is being accessed as you browse the methods in a class.

For this reason, it is imperative that you do not edit or modify the change log. You can view it with the Disk Browser and you can reinstall methods and class definitions from it into your system. This capability can be used to facilitate recovery from a system crash and is explained in more detail in the section on Surviving a System Crash in this chapter.

Other important events are automatically logged by Methods. Every time the image is saved, a message along with the date and time is written to the change log. Every Smalltalk expression that you evaluate with either do it or show it is also logged. In addition, everytime you remove a method from a class, a message is logged.

The format of the change log is very similar to the one described by Glenn Krasner in Chapter 3 of "Smalltalk-80: Bits of History and Words of Advice", Addison-Wesley c1983. A brief description of this format using the EBNF notation described in the Smalltalk Language Guide follows.

```
<rule>  ChangeLog = {expressionChunk | sourceChunk}.
<rule>  expressionChunk = evaluateChunk | imageChunk |
                          defineChunk.
<rule>  evaluateChunk = '"evaluate"' evalutatedText "!".
<rule>  imageChunk = '"*** saved image on: ' dateAndTime
                     '*****" "!"'.
<rule>  defineChunk = '"define class"' classDefinition "!".
<rule>  sourceChunk = "!" {className | className " class"}
                       " methods " "!" {method "!"} " !".
<rule>  evaluatedText is the text that was evaluated with
            all occurances of "!" replaced by "!!".
<rule>  dateAndTime is the date and time the image was saved.
<rule>  classDefinition is a valid class definition message
            as defined in the Smalltalk Language Guide.
<rule>  className is the name of a class.
```

<rule> method is a valid method as defined in the Smalltalk Language Guide with all occurances of "!" replaced by "!!".

The change log is a normal ascii text file which means that you can print it at any time. A simple examination of the file should remove any questions regarding its format.

Importance of Saving the Image

Methods is written entirely in Smalltalk and Smalltalk is a language of objects. Every piece of code and data in Smalltalk is a part of some object. The image that is referred to in this manual is the collection of all Smalltalk objects that make up the environment.

When you save an image, descriptions of all of these objects are written out on the image file. When you start the system at a later time, all of the objects described in the image file are recreated exactly as they were at the time the image was saved. Since these objects contain everything that makes up the Methods environment, the system starts up exactly as you saved it.

As you use the system, writing Smalltalk code or perhaps editing something else, changes are being made in memory to the objects and the object contents that make up the system. You must save the image to make your changes permanent. You can save the image by either selecting the save image function from the system menu or by selecting the save image function as you are exiting the environment. If you do not save the image, then the environment, the next time it is started, will not have any of your changes. The changes are not completely lost, however. Things that were logged in the change log are still in the change log and you can install them again using the Disk Browser (see the install function under "Editing Contents" in Chapter 3 and "Surviving a System Crash" in this chapter). Not saving the image is useful when you are developing something new and you decide that you have made a major mistake.

Compressing the Change Log

Compressing the change log reduces it to only one copy of each new or changed method. A new image is written automatically.

You should do this when the change log starts to get large. There must be enough space on the disk for both the new and the old change log at the same time, so you should not wait to the last minute.

To compress the change log, evaluate the following expression in any text pane (the System Transcript for example):

```
Smalltalk compressChanges
```

If you wait too long and there is not enough room on the disk, delete some files and then try it again. If you are using only floppies, then the best way to recover is as follows:

1. Save your image and exit Methods
2. Make a copy of the entire Image diskette
3. Restart Methods
4. Using the Disk Browser, remove the image file
5. Try compressing the changes again

This will work because the image file fills over half the diskette.

After you become familar with the Disk Browser, you should be able to figure out a way to do the above without exiting Methods.

Compressing the Source File

Compressing the source file creates a new source file for all of the methods currently in the system. Removed methods are deleted and the new or modified methods source is taken from the change log. A new image is automatically written.

To compress the sources, evaluate the following expression in any text pane (the System Transcript, for example):

```
Smalltalk compressSources
```

The source file (sources.sml) is written in a compressed format to preserve space on the disk. There are also pointers from the compiled methods in the image to the source file. For these reasons, it is imperative that the source file not be edited or altered except by Methods. It is perfectly okay to copy it.

Since there are pointers from the image to the source file, the image must always be used with the same source file (or a copy of it). After you compress the sources, you will have a new image file and a new source file. The change log will be empty. Therefore, it is a good idea to make a backup copy of the source file, the change log file, and the image file before compressing the sources. This way if anything goes wrong, you can recover.

After compressing the sources, you have a new base system. You should make a backup copy of the new source file, image, and change log. This will make it possible to recover from most, if not all, later disasters that might occur.

Surviving a System Crash

The automatic logging feature of Methods makes it possible to recover from most disasters. Disasters happen not because Methods has bugs in it, but because it is a modifiable program development environment. Methods will let you change anything, even if it destroys the environment. This presents no problem since you can recover most, if not all, of your work very quickly. In fact, experimenting with the system is a good way to learn about it, if you take the precautions described below.

1. Always have a backup copy of the source file that you are currently using. Methods never modifies the source file and neither should you. When you compress the sources, as explained in "Compressing the Source File", a new source file is made. Make a backup copy of it immediately.

2. Always have a reliable backup copy of the image file and change log file. Remember that the image and the change log are paired. You cannot use the image with an older version of the change log. After you compress the changes (see "Compressing the Change Log"), the image and change log files are both replaced. You cannot use an old image file with a new compressed change log file. You can use an old image with a later version of the same change log if it has not been compressed between the time when the image was made and when you use the change log. The simplest solution, rather than trying to understand the above rule completely, is to backup the image and change log files as a pair (that is why on a floppy system they are on the same diskette).

3. If you have been experimenting with Methods and you have the slightest reason to believe you have damaged the system or if there are changes you do not want to make permanent, do not save the image. Exit the Methods Environment using the forget image function. Remember, most of the changes you have made are in the change log and they can be reinstalled selectively.

4. If Methods crashes, don't panic. Make a copy of your Image disk. Most likely you can recover your work, and by making a copy you are able to try repeatedly. Read the next section for information on how to recover.

Recovering From a System Crash

If the system crashes you should make a copy of your Image disk before proceeding. To recover your work, follow these steps:

1. See if you have a good image file on the Image disk that he system crashed with. To find out, try to start Methods in the usual way using this disk. If it does not start up, then you should get your most recent back up of the image and change log and start Methods with that version.

2. Now that you have Methods running, use the Disk Browser to look at the change log file that you were using when the system crashed. The change log has all of the modifications that you made and all of the expressions that you evaluated.

3. Find the point in the change log where you saved the image that you are currently running. Every time that the image is saved, Methods writes a comment into the change log giving the time and date when the image was saved. It is usually easier to scan the change log backwards, especially since you are using a recently saved image.

4. Now that you have found the point at which you saved the image that you are now using, all of your lost work is from this point to the end of the change log. Remember that the change log is formated as a series of chunks. The exact format is given in the section "Automatic Logging of Changes" in this chapter. To restore something select one or more consecutive chunks of text. You must select complete chunks, watch the "!'s". Then pop up the pane menu and select the install function. If a chunk is an expression, it will be evaluated . If a chunk is a method definition, it will be recompiled and installed into the system. Choose the chunks that you install carefully, one of them represents the error that you made and you do not want to repeat it.

When you have finished recovering what you want to, save the image.