Using the Apple Dylan Development Environment

Preliminary

Developer Press
Apple Computer, Inc.

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Contents

Preface **Preface** vii

What to Read vii
Conventions Used in This Book viii
  Special Fonts viii
  Types of Notes viii
For More Information ix

Chapter 1 **Learning Apple Dylan** 1

Introducing the Apple Dylan development environment 1
Running Apple Dylan 8
  Running the development environment 9
  Opening a sample project 11
  Running a sample application 15
Using browsers 15
  Browsing a project 16
  Using the built-in browsers 24
  Linking panes and browsers 34
  Showing different aspects of objects 42
  Changing aspects 46
  Using the browser References To 49
  Using the browser Info for Selected Class 53
  Customizing browsers 58
  Saving a browser configuration 61
Editing in Apple Dylan 66
  Copy, Cut and Paste in Apple Dylan 67
    Copy Special and Insert Special commands 68
  Undo and Clear commands 69
  Replace and Find commands 69
  Formatting commands 70
  Importing and exporting Dylan text files 70
  Macintosh and Emacs-style editing commands 71

Macintosh-style key commands 71
Emacs-style key commands 72
Editing code 76
Customizing the development environment 83
Setting development environment preferences 84
Setting editing defaults 85
Setting Listener interaction defaults 87
Using icons in Apple Dylan 88

Chapter 2 Using Apple Dylan 95

Apple Dylan User Model 95
The Project 95
What goes into a project? 97
What's *really* in a project? 101
What happens when you open a project? 101
The active project 102
Targeting 68K and PowerPC Platforms 103
Application or library? 104
Library version numbers 106
The Application Nub 107
Keeping your project synchronized 109
Orphan definitions in the runtime 111
Status indicators and synchronization 111
Restoring synchronization 112
Apple Dylan Listener 113
Inspector windows 116
Bailing out of Apple Dylan 117
Building standalone applications 118
Starting a project 119
Creating a new project 120
Setting the project type for an application 126
Setting the project type for a library 128
Saving a project 130
Saving the objects in a pane 130
Adding a subproject or resource file 130

Adding the framework 131
Including C code 132
Compiling your project 133
Launching the runtime 135
Untethering from the runtime 136
Checking code status 136
Compiling a selection 139
Compiling all uncompiled code 140
Compiling code from the Listener 141
Excluding code from compilation 143
Including code in compilation 143
Running an application in Apple Dylan 144
Tethering to a running application 145
Debugging a project 146
Inspecting the stack 148
Inspecting Listener results 151
Inspecting heaps 153
Inspecting modules 154
Metering expressions 156
Monitoring an individual function 159
Creating a user interface 161
Adding the Apple Dylan interface builder 162
Adding the new user interface 163
Sharing your user interface 164
Building your application or library 164
Building your standalone application 164
Building a library 166
Sharing code 167
Sharing projects 168
Sharing code by exporting 168
Retrieving code by importing 171

Chapter 3 Apple Dylan Reference 173

Key Command Shortcuts 173
Command Reference 175

Stack window commands 222
Inspector window commands 223

Glossary 273

Index 279

Preface

This book, *Using the Apple Dylan Development Environment,* gives you a good look at the user interface and program development environment in Apple Dylan. You'll find that the Apple Dylan tools allow flexible programming and full access to information about your program.

The term **Apple Dylan** refers to the development environment and associated tools, extensions and capabilities developed by Apple Computer for programming in the **Dylan** language, an **object-oriented dynamic language**.

For more information on programming in the Apple Dylan language, see the books *Programming in Apple Dylan* and *Apple Dylan Extensions and Framework Reference*. For more details on creating a user interface for your application, see the book *Creating a User Interface in Apple Dylan*. For information on how to install and configure Apple Dylan on your system, see the booklet *Apple Dylan Quickstart*.

What to Read

This book has three chapters, a glossary, and an index. You can either read it sequentially or move around in it from one topic to another.

- Learning Apple Dylan—introductions to and explorations of the development environment. This chapter should give you a chance to learn how to interact with the development environment without a lot of explanation. This chapter primarily familiarizes you with the browsing and editing features of the Apple Dylan environment.
- Using Apple Dylan—deeper understanding and in-depth tasks described. This chapter opens with a description of the Apple Dylan User Model, which is the chain of actions and events that takes you from starting to write code, through compiling, and finally to creating a running standalone application.

- Apple Dylan Reference—alphabetical listing of all commands used in Apple Dylan, along with examples. This chapter opens with a list of the keyboard equivalent shortcuts for each command.
- Apple Dylan Glossary—all words introduced in **bold** are defined in the glossary, along with many other terms used in Apple Dylan.

All chapters include many examples, and Learning Apple Dylan and Using Apple Dylan include many step-by-step descriptions of common tasks.

Apple Dylan is probably different from any development environment you have used in the past. By following the information presented in this book you will soon find yourself interacting with the programs you develop in an entirely different way, not as an editor buffer full of code or as a flow chart, but as a living program. All of the program's features are directly under your control and organized, not by the requirements of a language or compiler, but according to the way the program actually works.

Conventions Used in This Book

This book uses various conventions to present certain types of information.

Special Fonts

All code listings, reserved words, and the names of data structures, constants, fields, parameters, and functions are shown in a monospaced font (`this is monospaced`).

When new terms are introduced, they are in **boldface.** These terms are defined in the glossary.

Types of Notes

There are several types of notes used in this book.

Note
A note like this contains information that is interesting but possibly not essential to an understanding of the main text. Often, these Notes include additional information. See for instance,"The active project" on page 102.

▲ **WARNING**
A note like this contains information that is especially important. As this is an early release of Apple Dylan, not all interactions are as smooth as we want them to be. For instance, the warning in "The active project" on page 102. ▲

For More Information

APDA is Apple's worldwide source for hundreds of development tools, technical resources, training products, and information for anyone interested in developing applications on Apple platforms. Customers receive the *APDA Tools Catalog* featuring all current versions of Apple development tools and the most popular third-party development tools. APDA offers convenient payment and shipping options, including site licensing.

To order products or to request a complimentary copy of the *APDA Tools Catalog*, contact

APDA
Apple Computer, Inc.
P.O. Box 319
Buffalo, NY 14207-0319

Telephone	1-800-282-2732 (United States) 1-800-637-0029 (Canada) 716-871-6555 (International)
Fax	716-871-6511
AppleLink	APDA

America Online	APDAorder
CompuServe	76666,2405
Internet	APDA@applelink.apple.com

CHAPTER 1

Learning Apple Dylan

Dylan is a new programming language and Apple Dylan is a development environment designed to be used with Dylan. The language provides automatic memory management, type checking at both compile-time and run-time, and a high-level exception handling mechanism. The Apple Dylan framework further simplifies programming.

The Apple Dylan development environment supports incremental compilation and the Apple Dylan Listener for executing individual expressions without having to create the code scaffolding around them. User-interface design can be performed in the graphically-oriented user-interface builder, which allows you to see what your user interface will look like before you have written any code for it, thus cutting down on recoding.

This chapter provides a general introduction to Dylan and the Apple Dylan development environment, including installation, examples of interacting with the browsers, and editing. See the chapter "Using Apple Dylan" on page 95 for more specific information on using the environment to develop applications.

Introducing the Apple Dylan development environment

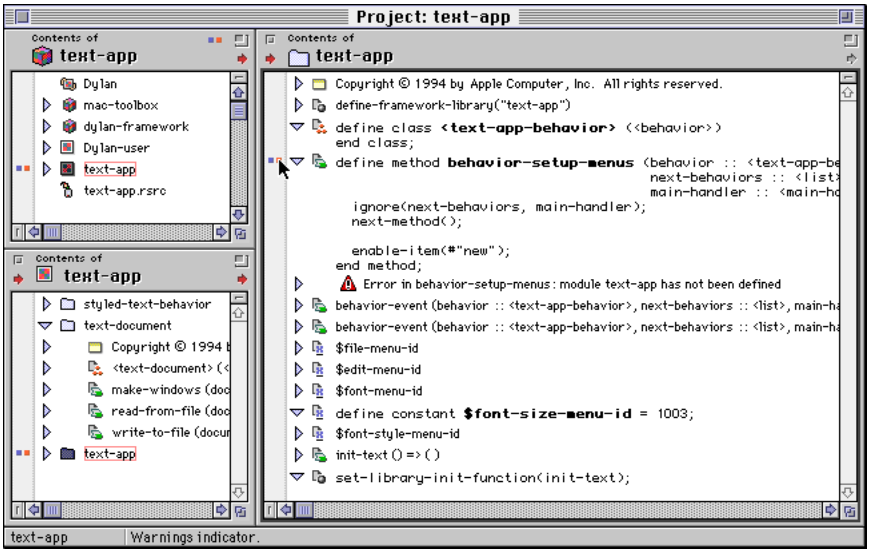
In Apple Dylan, you write code, compile, debug and build an **application**, but the way you go about it is probably different from what you are used to. For instance, the source code for an application is not written in files, but is broken into individual **source records**, each method and class in its own source record. Each construct of the Dylan language is an object that is stored individually in a database and can be manipulated independently making the concept of header files and object files obsolete. Everything that you have compiled is stored in a **database** that can be viewed and manipulated through Apple Dylan browsers.

The **source** database contains all your source code with all its links and structure. What you see is a **project** displaying objects of finer and finer granularity, right down to the source code. The largest containers in a project are modules, which contain source folders. The source folders hold source records, which contain the source code. All of these containers can be dragged, dropped, and copied as if you were in the Finder.

The development environment displays the project in **browsers**, windows with one or more linked **panes**. The browsers operate as if a set of panes were tiled next to each other and then bound together so they could interact. Most browsers are linked together in such a way that the object you select in one pane has its contents displayed in another pane. This way you can browse through a list of objects in the first pane, examining the contents of each in turn. Information displayed in each pane is under your control.

A number of browsers are supplied with Apple Dylan but you can also create browsers of your own. The main browser for a project is called the **project browser**. The browser in the following figure is the **default project browser** shipped with Apple Dylan. When you first open or create a project, this browser configuration is displayed. In this example, the sample project text-app was opened.

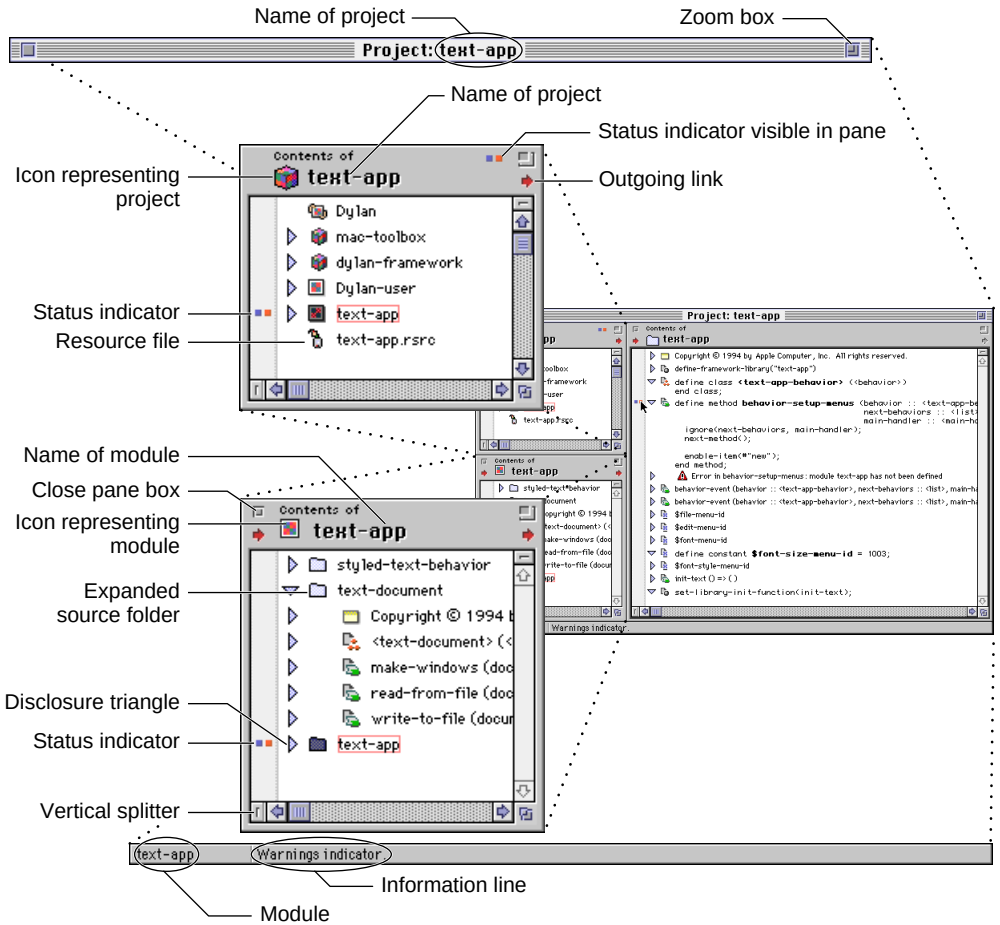
The default project browser has three panes linked to display the text-app project. The title bar for the browser displays the project's name, as does the header of the pane in the upper-left. That pane displays the major components of the project, mostly modules and subprojects. This is the **root pane** of the browser because it is the browser's upper-left pane. The root pane's **basis** (or source) is the project itself.



The pane in the lower-left displays the contents of whatever object you select in the upper-left pane. The panes have been linked in the project browser to display that relationship, although that **link** could be changed to display other relationships. Likewise, in the default project browser the contents of any object selected in the lower-left pane are displayed in the pane on the right.

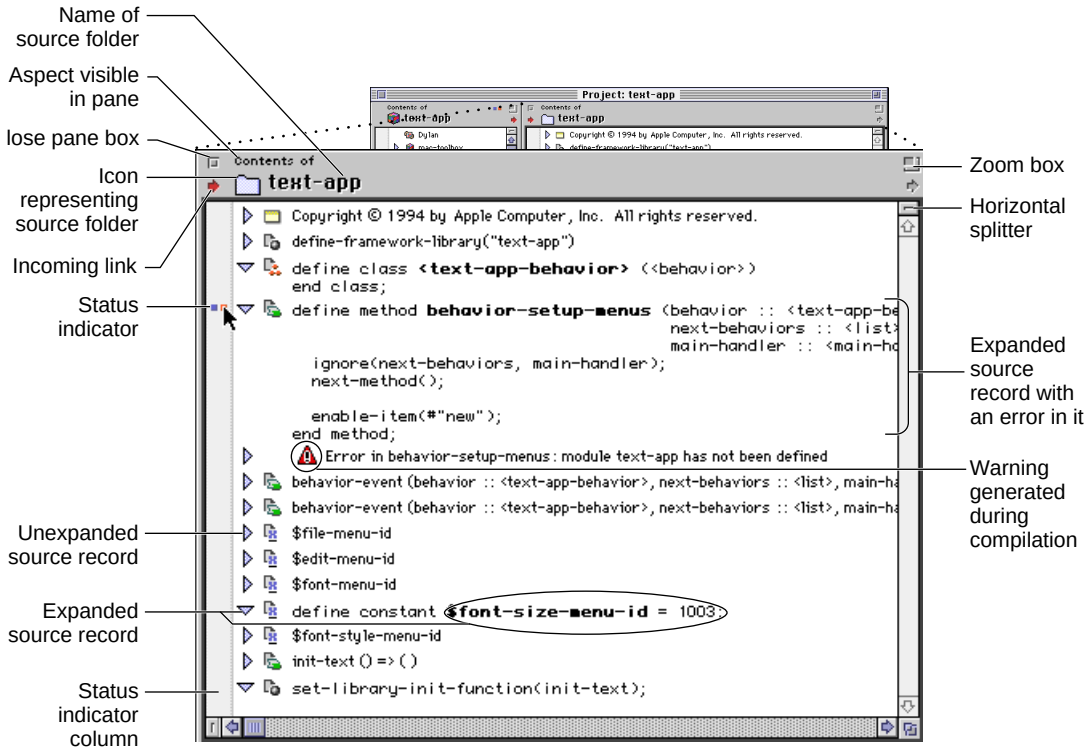
In the previous figure the module named text-app has been selected in the upper-left pane. That module holds the code for the project that the project's author wrote. Its contents appear in the lower-left pane. Notice that some of the contents have been expanded within the pane using **disclosure triangles**, just as you would in the Finder. The last object in the lower-left pane, a source folder named text-app, has been selected, so its contents appear in the pane on the right. In the pane on the right the numerous source records are listed and some of them have been expanded. There is even a warning that has been generated during compilation. The warning appears at the end of the source code for the source record behavior-setup-menus.

The following figure shows the two left-hand panes of the browser in the previous figure. This figure identifies some of the features on the panes and on the browser, which surrounds the panes. The browser's name is on the bar at the top of the figure. In this example, the name is "Project: text-app".



The **information line** prompts you with ephemeral information depending on what you have selected. If you have selected a function, its arguments are displayed. If you are dragging an object, you are told whether it can be dropped or not. In general, the information line gives feedback on current activities in the browser; it is sometimes called the prompt area or status line. The information line may be blank. The current module is displayed to its left.

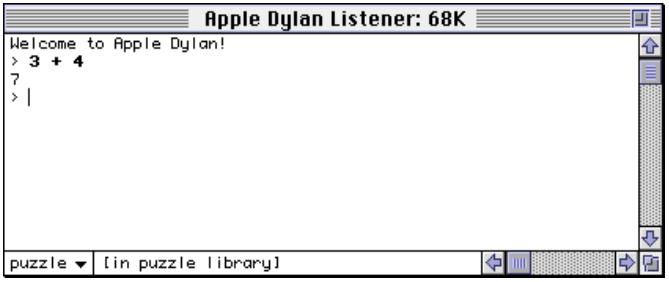
The following figure identifies the features of the pane on the right side of the browser.



There are many other types of browsers provided or supported by the development environment to help you understand your project. You can create a browser based on any **aspect** of an object. For instance, you could get a list of the family of methods and the generic function a method belongs to. In addition to the browsers supplied with Apple Dylan, you can create your own, suited to the needs and nature of your project.

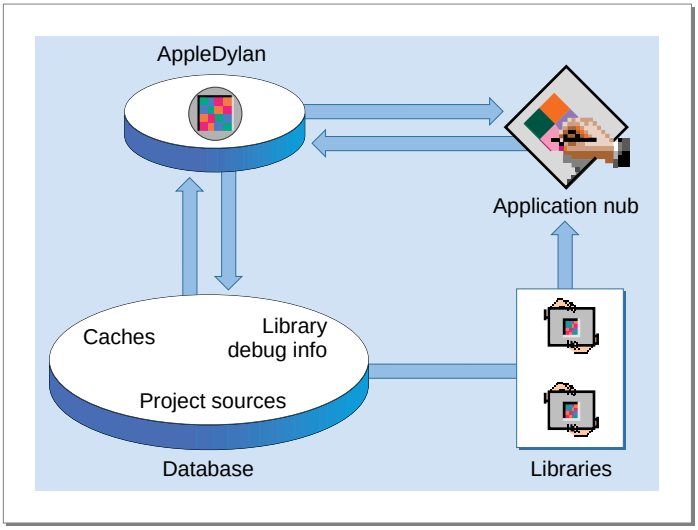
The development environment provides a means of executing individual **expressions** outside the browsers. The Apple Dylan Listener allows you to type in an expression and execute it independently of the project's structure of **containers**. The Listener allows you to try out coding ideas without creating an entire structure to hold the code. In addition, all values returned and other program results are printed to the Listener window for any code executed anywhere in the development environment, even in the browsers.

You can enter any Dylan expression into the Listener for immediate execution. For example, you can call functions in the Listener (including your application's startup function), and you can define functions, variables, and classes in the Listener. The following figure shows the Apple Dylan Listener window with a simple expression, 3 plus 4, entered at the prompt and the return value, 7, returned by the Listener. The Listener is ready for the next expression, as indicated by the waiting prompt, >.



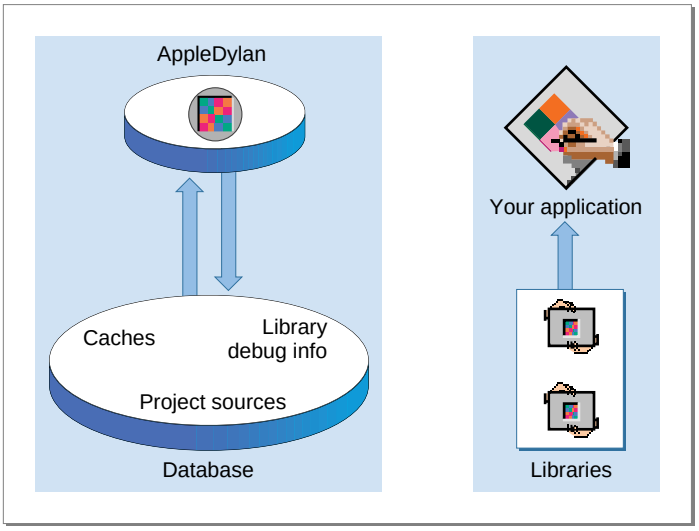
Your application under development can run under the control of the Apple Dylan development environment. It doesn't need to be complete and built as a standalone application to do so. While you are still developing the program, your code can run loaded into a small, "nub" of an application called the **Application Nub,** which is tethered to the development environment. The code you write is added to the application nub every time you compile the code, allowing you to slowly build up to your final standalone application.

The following figure shows Apple Dylan tethered to the Application Nub, as it is while you are developing your application.



When your project is complete and you build your standalone application, the services provided by the application nub and the code in your project are bundled together. The connection to the development environment is severed.

The following figure shows that when you build your standalone application, the Application Nub and your application have become one and it has been severed from the development environment.



This means that from your point of view as a programmer, the application under development and the standalone application are equal. Just as you can interact with an unfinished project connected to the application nub, so too can you interact with a standalone application that was already been built in Apple Dylan. You might need to do this for further debugging of the application.

The browsers and the Listener help you write your source code and compile it, but you need to debug it too. For that the development environment provides inspector windows. When you have a problem, inspector windows let you examine runtime objects to help you determine the cause of the problem.

Running Apple Dylan

When you have your system configured, you can install Apple Dylan by following the instructions in the pamphlet *Apple Dylan Quickstart*.

To use Apple Dylan you must run it and open a project, either a sample project or a new one. You might also want to run some of the sample projects' standalone applications in the Finder to see what they do. You can run the interface builder's standalone version from the Finder if you want to see how it works. To use it as you write your application, run it from inside the

development environment. See the chapter “Using Apple Dylan” on page 95 for more information.

You can run Apple Dylan on two machines by installing the Application Nub and any pre-built libraries on a separate machine from the rest of the development environment. The pre-built libraries, such as the Dylan library or any libraries you have included with your application, must be in the same folder as the Application Nub or in the Extensions folder on the separate machine. If you wish to do two-machine development, see the book *Apple Dylan Quickstart* for more information on memory and special setup requirements.

Running the development environment

Apple Dylan is an application in the Apple Dylan folder you copied from the Apple Dylan CD to your computer. You can run Apple Dylan SI instead, if don’t have enough RAM to run the standard Apple Dylan. Depending on the amount of RAM available on your system, you might want to quit any other applications before running Apple Dylan.

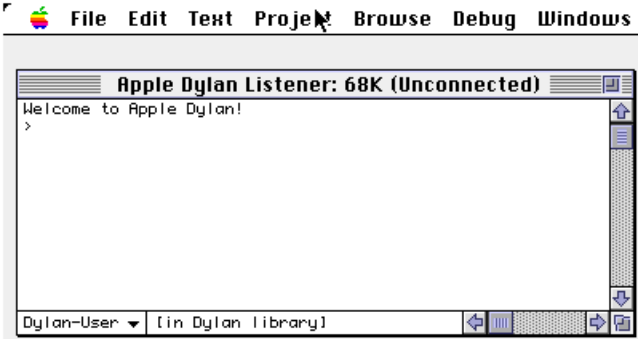
Ordinarily, to run Apple Dylan you simply double-click its executable file. In the following task you also adjust the memory. That is something you only need to do the first time you run Apple Dylan.

You can run Apple Dylan on two machines by installing the Application Nub and any pre-built libraries on a separate machine from the rest of the development environment. The pre-built libraries, such as the Dylan library or any libraries you have included with your application, must be in the same folder as the Application Nub or in the Extensions folder on the separate machine. If you wish to do two-machine development, be sure to select that on the Application Nub sheet of the Preferences command. See the book *Apple Dylan Quickstart* for more information on memory and special setup requirements for two-machine development.

1. **Follow the instructions in *Apple Dylan Quickstart* to set up your machine properly.**
2. **Open the Apple Dylan folder.**
   The Apple Dylan folder contains two main folders, Apple Dylan and Apple Dylan documentation. The Apple Dylan applications are in the folder Apple Dylan.

3. **Double-click on the Apple Dylan application.**
   You can also run Apple Dylan by double-clicking on Apple Dylan SI.

   When you run Apple Dylan, the development environment displays the unconnected Listener and the Apple Dylan menu bar. To use Apple Dylan you should open a project. For more information on that, see the next task.



4. **Double-click on the file Application Nub from the Finder.**
   The Application Nub file is in the folder Apple Dylan: Apple Dylan Files: Application Nub.

   You must set the preferred memory for both the Application Nub and Apple Dylan files to leave at least 50K unused when both are running. This is because the system heap in Apple Dylan cannot expand properly when all available memory has been allocated to applications. When you run Apple Dylan in the future, after resetting the preferred size for these two files, you don't need to run the Application Nub file as well.

   You will notice that the Application Nub has no user interface within Apple Dylan, so even when it's running, there is no menubar for it. That is because it is the nub, or core of, the application you create. You create the menubar and other user interface elements in the course of creating your application. Ordinarily you launch and quit the Application Nub from within Apple Dylan.

5. **Quit Apple Dylan and choose About This Macintosh from the Apple menu.**
   Note how much unused memory is left when both applications are running.

6. **Set the preferred size for both Apple Dylan and the Application Nub using the Get Info command.**

7. **Double-click on the Apple Dylan icon from the Finder and then double-click on the Application Nub.**
8. **Choose About This Macintosh from the Apple menu.**
   Verify that at least 50K of unused memory is available.
9. **Double-click on Quit Application Nub from the Finder.**
   When you have the memory adjusted properly, quit the Application Nub before proceeding with any other tasks in this chapter. The Quit Application Nub file is in the folder Apple Dylan: Apple Dylan Files: Application Nub.

   Ordinarily you launch and quit the Application Nub from within Apple Dylan. However, in this case and other rare cases, you might need to quit the Application Nub using this executable file in the Finder.

Opening a sample project

Several sample projects have been included with Apple Dylan. You can find the samples in the Sample Code subfolder or via an alias to more samples that is in Sample Code. To open a project, you use the Open command on the File menu and select its project file, which has the suffix ".π". You can also open a project by double-clicking on its project file in the Finder. This will also launch Apple Dylan if it's not already running.

Open one, such as the puzzle project, to see how it looks in the development environment. The sample project puzzle is interesting because it includes the Apple Dylan framework, which is a subproject called dylan-framework in the root (upper-left) pane of the project browser. It also includes mac-toolbox, which allows access to existing Macintosh toolbox code.

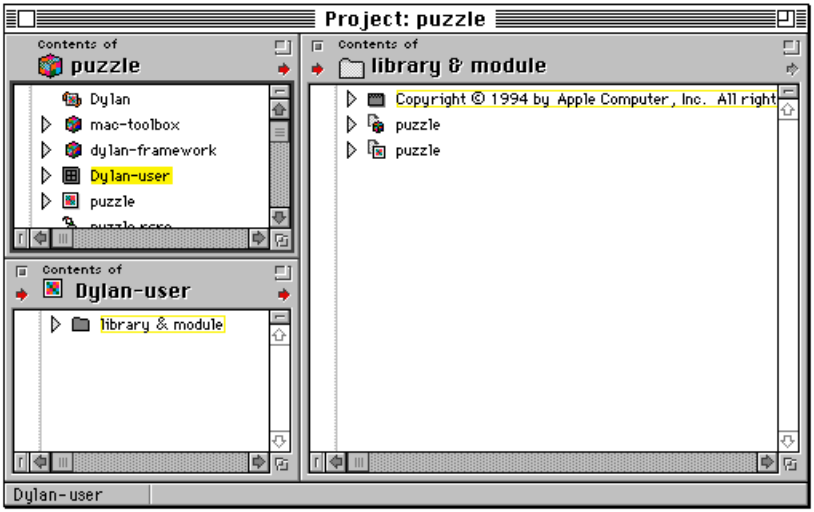
The first project you open in each session of Apple Dylan is automatically made the **active project**. If you open other projects in the same session, they are inactive. Only one project can be active at one time and only the active project has full Apple Dylan functionality in it, such as being able to compile it or see its various aspects. You can change the active project using the Activate Project command.

1. **Choose Open from the File menu.**
   A dialog box allows you to choose the sample project you want to open.

2. **Choose the alias "More Samples" to go to more sample code.**
   In the folder Sample Code in Apple Dylan, you will find several sample projects, as well as an alias to other samples. The alias takes you to samples that use the framework.

3. **Find the puzzle project, which is in the puzzle folder, and select the file that ends with ".π".**
   Click OK when you have selected the file puzzle.π.

   The puzzle project opens displaying its default project browser. It consists of three panes. The pane in the upper-left is the root pane for the project browser. You can see the name of the project in the browser's name, "Project: puzzle," as well as in the header of the root pane. The icon for a project is also in the root pane's header.



4. **Resize the project browser for easier viewing, if you want.**
   You can expand the browser, if you want, by zooming the window with its zoom box. You can also drag the browser and each pane with their resizing boxes in their bottom right-hand corners.

   Notice the six objects in the root pane. Three are libraries, which are represented with the subproject icon in Apple Dylan, two are modules and one is a resource file.

Dylan is a library that contains the basic definitions of the Dylan language and the Apple Dylan extensions. This subproject is automatically included in all projects and cannot be removed or modified.

Mac-toolbox is a library containing the Macintosh toolbox calls that have been included with this project. It is up to you to add this library and include any of the Macintosh toolbox calls you want to use in your project.

Dylan-framework is a library containing the Apple Dylan application framework. You must add the framework to your project if you want to use it; it is not automatically added to a new project. Since the source code for the framework is included in Apple Dylan, you could modify it if you want, however, this is not a common practice, nor is it recommended.

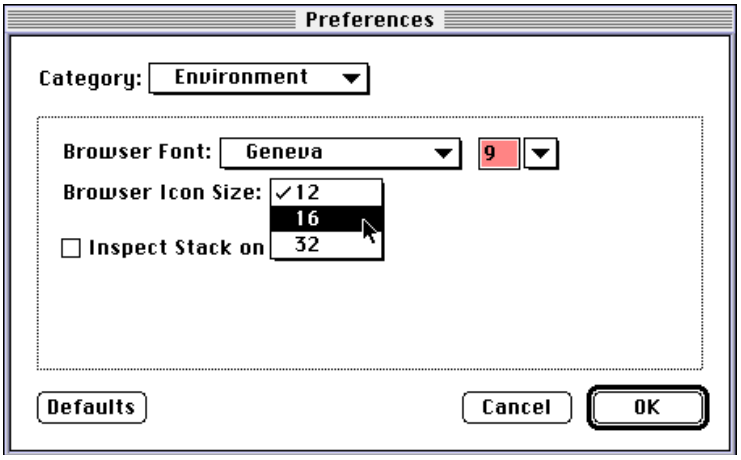
Dylan-user is a module. Every project has its own individual Dylan-user module. In very simple projects all the code for the project can be in the Dylan-user module. More commonly, it is standard practice for the Dylan-user module to contain the module definition and library definition used to structure the rest of the project. You will have to write your own module definition and library definition for your project.

The puzzle module is a container for most of the source code for this project. You will create your own module for your project that will contain your source code.

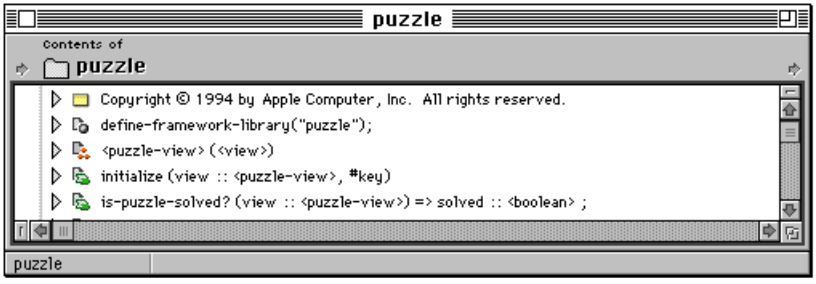
The file puzzle.rsrc is the resource file used by the puzzle project. If you want to use resource files in your project, you must add them to your project using the Add to Project command.

5. **Choose the Preferences command from the Edit menu.**

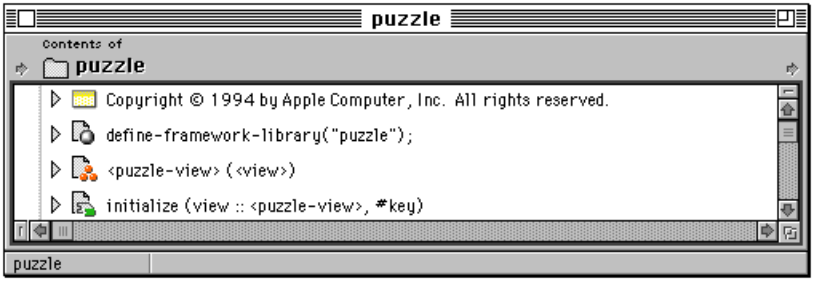
6. **Increase the icon size from the Environment sheet.**
   By changing the Browser Icon Size setting to 16 (or even 32), the differences between icons are easier to see. If you do that, you might want to increase the font size to 10 or more as well. For a complete listing of the icons in Apple Dylan, see the section “Using icons in Apple Dylan” on page 88. You can set other preferences from the other Preferences sheets. They can be reached from the Category pull-down list at the top of the Preferences dialog box.



The following figure shows part of a pane with the icon size at 12, which is the default.



The following figure shows part of a pane with the icon size increased to 16.

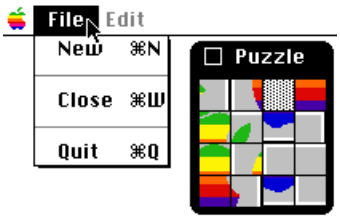


Running a sample application

You can run the sample applications from the Finder to see what they do as standalone applications. To run applications from inside Apple Dylan, use the Run command; see the section "Running an application in Apple Dylan" on page 144.

1. **Open the Apple Dylan folder, if it's not already open.**
   The Dylan Files folder contains several folders, including Sample Code, which contains the sample projects and their standalone applications. It also contains an alias to samples that use the framework.
2. **Open the Sample Code folder and choose the alias to more samples.**
   This takes you to samples that use the framework.
3. **Double-click on a sample application, such as the puzzle application.**
   The application's executable file appears just as all executables files do on the Macintosh.

   The following figure shows the puzzle application running. The game is to move the squares on the puzzle around until an image appears. The puzzle application has a menu bar that allows you to create new puzzles and to quit. When you're done observing the sample application's behavior, quit the application.



Using browsers

Browsers are windows with one or more panes. The panes are tiled so none are in the background or foreground in relationship to one other and can be linked to work together. Browsers are where you will do most of your coding and compiling.

If you have just launched the development environment, you need to open a project before you can open any browsers. The project browser must remain open as long as you are working on that project. To close the project browser is to close the project.

Once you have opened a project or created a new one, you can open other browsers to get other views of the project. Several built-in browsers are listed in the third and fourth sections of the Browse menu. To get a complete list of all browsers available to you, choose List of Browsers from the Browse menu. You can have several browsers other than the project browser open at once and closing them does not close the project.

You can change the configuration of the panes in an existing browser, create new browsers, and save a browser's configuration. You don't need to save the code changes you make in a browser when you close it, but if you have changed the configuration of the browser, you can choose to save that new configuration with the Save Browser command.

The code changes are not lost when you close a browser (unless that browser was the project browser and you specifically chose not to save any changes). To save code changes, you select the pane containing the changes, and then choose Save from the File menu. Likewise, you can save all the code in an entire project by issuing a Save All command or simply choosing to save it when prompted to do so as you close its project browser.

You can open a project from inside the development environment or from the Finder. Double-clicking on a project file, which has the suffix ".π", launches Apple Dylan (if it's not already running) and opens that project.

The sample projects provided with Apple Dylan include puzzle, text-app, tiles, paint-app, hello, Online Insultant, and skeleton, among others. The sample projects can be found in the folder Sample Code, or the alias in it. Some have a Read Me file with them, either as a file in their folder or within their project itself as a text file.

Browsing a project

When you open a project, its project browser appears. In the upper-left pane, or root pane, of that browser the contents of the project are listed.

Every project has a Dylan subproject, which always appears first on the list in the root pane of the project browser. This is the library containing the Dylan

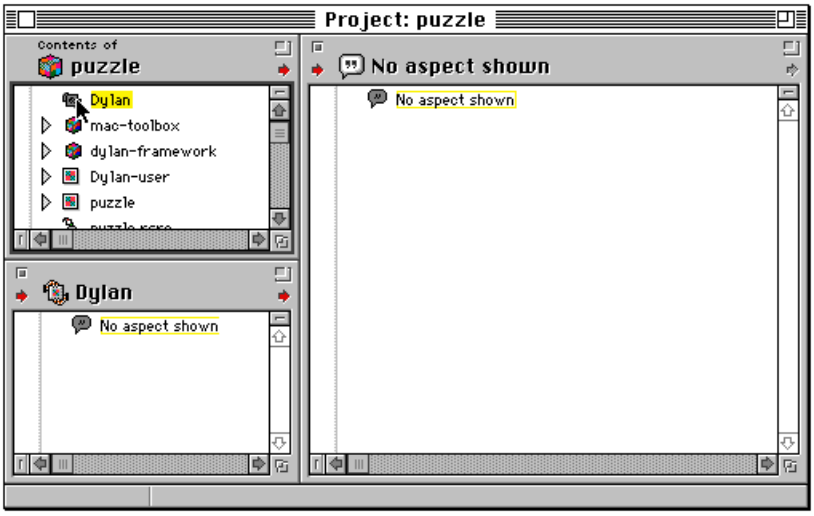
language itself. The Dylan subproject is created automatically and you cannot edit it.

Every project also has a Dylan-user module. It is also created automatically when you create a new project, but it must be edited to specify your project's library and module definitions.

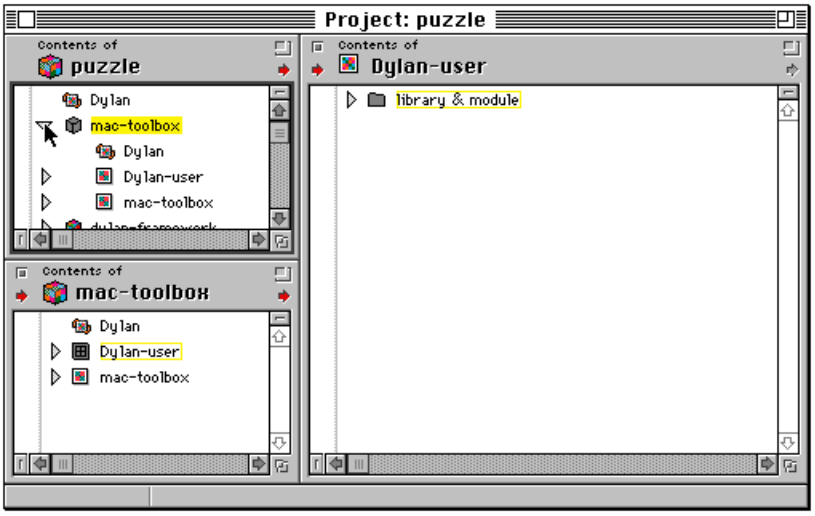
If a project uses the framework or the mac-toolbox library, these also appear as subprojects. You can add these to a project using the Add to Project command on the File menu.

The default project browser has three panes that are linked so they can work together to display a project in detail. Whether you click an object's disclosure triangle, icon, or name produces different results in a browser. Double-clicking produces still other results. You can change the aspect visible in a pane, if you wish, using the Aspect command on the Browse menu.

1. **Open the puzzle sample project, if it's not open already, and make sure it's the active project.**
   Use the Open command and choose the file puzzle.π. The sample projects can be found in the Sample Code folder or the alias in it. If you had another project open before opening puzzle, you will need to click in the puzzle project once you open it, and then use the Activate Project command from the Project menu. If puzzle were not the active project, the project browser's name would read "Project: puzzle (inactive)" and the root pane's name would read "puzzle (inactive)".
2. **When puzzle opens, click on the icon or name of the subproject Dylan.**
   You will see that you can't open it and a message prints "No applicable aspect". This subproject is the library containing the Dylan language, which you cannot change (or even view) from within the development environment.



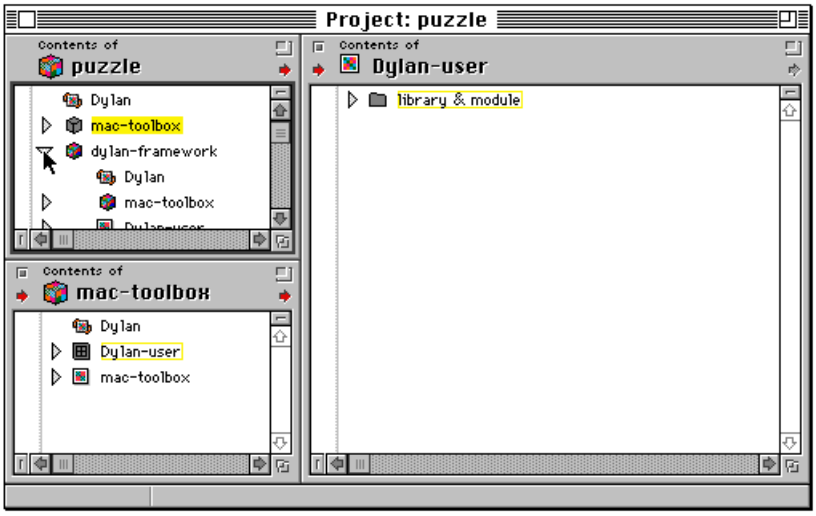
3. **Click the icon and then the disclosure triangle for the subproject mac-toolbox.**
   You can see the contents of an object by clicking on the disclosure triangle to its left. When you click its disclosure triangle, the contents of mac-toolbox are displayed much as they would be in the Finder. The contents are Dylan, Dylan-user, and mac-toolbox. These are references to the Dylan project, the Dylan-user module, and the mac-toolbox module.



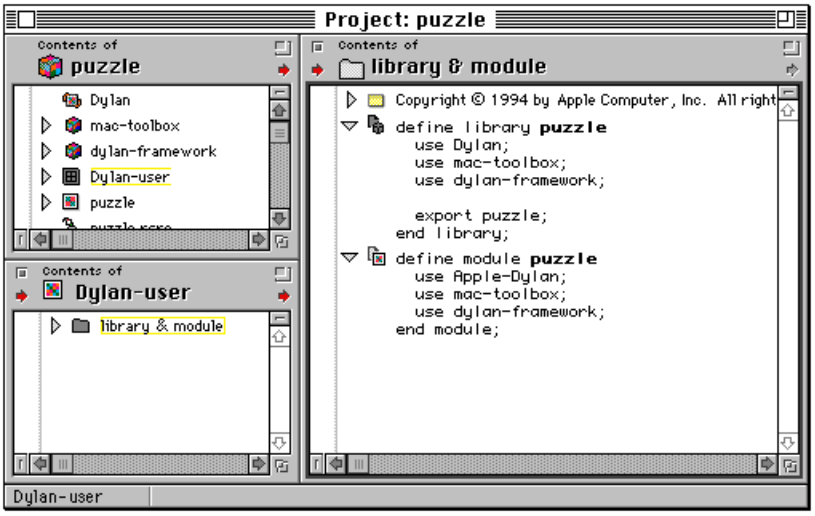
For now, click the disclosure triangle again to collapse the list of the contents of mac-toolbox.

4. **Click the disclosure triangle left of the subproject dylan-framework to expand its contents inline.**
   This is the Apple Dylan framework, which you might want to study later. Its contents are Dylan, mac-toolbox, Dylan-user, dylan-framework, and dylan-framework.rsrc. Don't be concerned about efficiency or memory and disk space when you see redundant references to such objects as Dylan or mac-toolbox. These are only references to these objects, which are not being duplicated with each reference.

   As shown in the following figure, since you did not select the name or icon of dylan-framework, only its disclosure triangle, you have not selected dylan-framework. The subproject mac-toolbox was your last selection so its contents are displayed in the other two panes in the project browser.



5. **Click the icon or name of the Dylan-user module in the root pane, then click the disclosure triangles for its library and module source records in the pane on the right.**
   As shown in the following figure, the module Dylan-user has been selected so its contents, the source folder library & module, appear in the lower-left pane. Then library & module has been selected so its contents appear in the pane on the right. This is because the three panes are linked in this way.

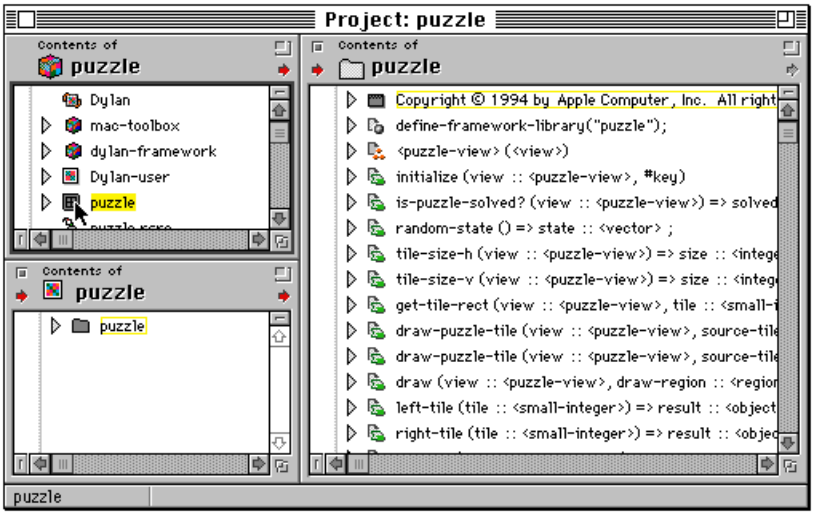


When you click the name or icon of an object in a pane, such as the root pane, you see its contents displayed in the pane it's linked to, the lower-left pane in this case. The name and icon for the selected object, in this case the Dylan-user module, also appear in the header of the other pane. As seen in the preceding figure, the header for the lower-left pane shows the Dylan-user module's name and icon, as well as its default aspect, Contents of.

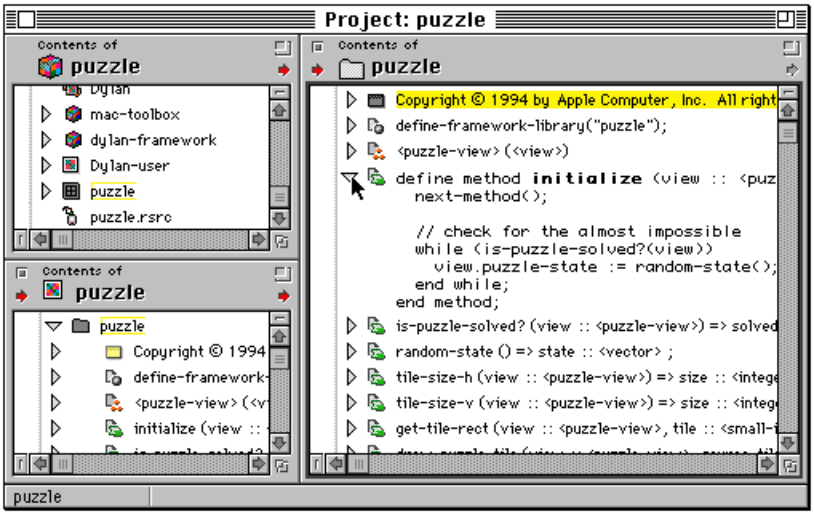
The lower-left pane is linked to the right-hand pane, so the contents of the object selected in the lower-left pane appear on the right. In this case, the contents of the library & module source folder in the lower-left pane appear in the right-hand pane. The contents of the source folder are the source records. The header for the pane on the right now has the name and icon for the library & module source folder in its header. Its default aspect is also Contents of.

Within library & module are the two source records, the puzzle library definition and puzzle module definition. As shown in the preceding figure, they have been expanded in the pane on the right to reveal their source code. Notice that a dark line surrounds the content area of the right-hand pane in the following figure, indicating that it is the active pane. When you click in a pane, you make it the active pane.

6. **Select the module puzzle in the root pane.**
   The puzzle module contains the source folder puzzle, which in turn contains numerous source records. The source records appear in the pane on the right.

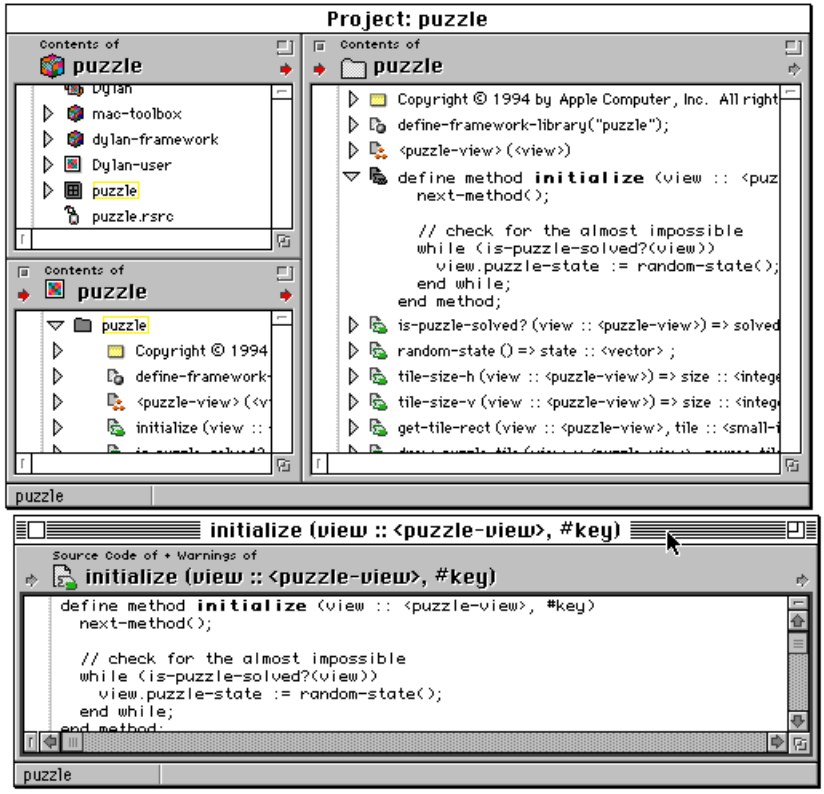


7. **Expand the contents of the puzzle source folder by clicking its disclosure triangle in the lower-left pane.**
   When you expand the puzzle source folder in the lower-left pane, you see its contents listed inline in the lower-left pane. If you select the puzzle source folder, its contents are also listed in the right-hand pane because these two panes have been linked. You could edit this code in either pane and the other pane would immediately update. For now, just click disclosure triangles in the right-hand pane to see the code in individual source records. You can click the pane's zoom box for more space or drag its resizing box to make it bigger.



8. **Double-click on any object, such as the source record initialize.**
   You can double-click on any object, which means its icon or name, to open a new, separate browser for it. In the following figure, the source record initialize has been expanded and it has also been double-clicked. Double-clicking on an object opens a new browser which, in the following figure, has been dragged below the project browser so you can see both browsers. You could change the source code in either browser and the other would be updated immediately.

   The default aspect of the double-clicked object determines which aspect of the object is displayed in the new browser. In this case, the default aspect for initialize is Source Code of +Warnings of. That is the default aspect for all source records.



9. **Double-click the object puzzle.rsrc in the root pane of the project browser.** You might need to scroll the root pane so you can see the resource file. You cannot edit a resource file from within Apple Dylan, but must use the resource editor of your choice from the Finder. You add a resource to a project using the Add to Project command.

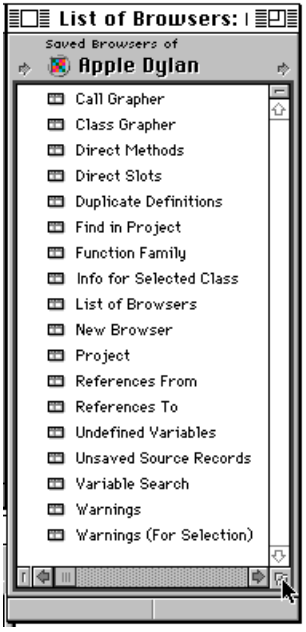
Using the built-in browsers

There are many other browsers in Apple Dylan in addition to the project browser and the single-paned, new browser described in the previous task. These built-in browsers display various aspects of the active project and its contents. Using these browsers, you can see such aspects as all the unsaved source records in the active project or see a graph of a class's inheritance

hierarchy. For more information on aspects, see “Showing different aspects of objects” on page 42 and “Changing aspects” on page 46.

The built-in browsers are available on the Browse menu. The browsers listed in the third section of the Browse menu work with the project as their bases, while those in the fourth section work with selected objects as their bases. You can open only one of each browser from the third section, but as many as you want from the fourth section, if you choose different objects as their bases. For example, the browser Unsaved Source Records from the third section lists all the unsaved source records for the entire active project. Only one such browser is necessary, so only one can be opened. By contrast, each of the browsers in the fourth section can have many versions open at once, since you could select any number of classes, for example, and open a Direct Methods browser for each.

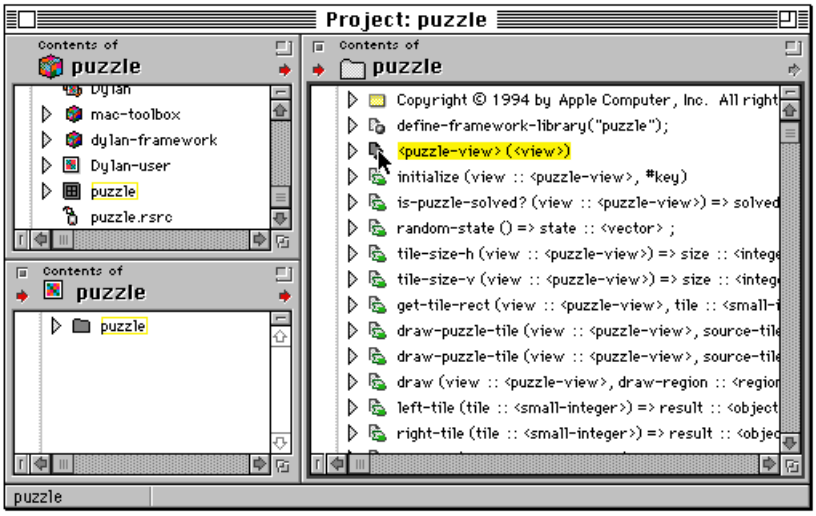
The List of Browsers browser in the third section lists all the browsers available and includes commands from the Browse menu. The objects on List of Browsers have drag and drop functionality, which makes it a handy browser to keep open as you are using Apple Dylan.



For more information on List of Browsers, see its reference entry in the third chapter.

1. **Open the puzzle sample project, if it's not already open, and make sure it is the active project.**
   The sample projects can be found in the Sample Code folder or an alias in it.

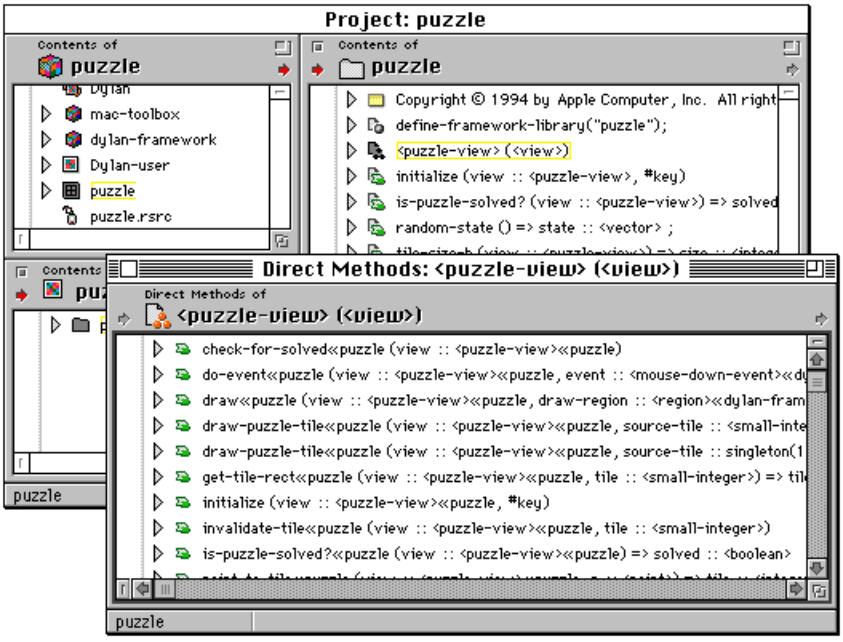
   In the following figure you can see that the module puzzle has been selected in the root pane. Its contents are listed in the lower-left pane. In addition to the puzzle module being highlighted in the root pane, it is also named at the bottom of the project browser as the active module. The puzzle source folder has been selected in the lower-left pane, thus revealing its source records in the right-hand pane.



2. **Select a source record for a class in the right-hand pane, such as the class `<puzzle-view>`, and choose Direct Methods from the Browse menu.**
   The class `<puzzle-view>` is the third source record from the top in the right-hand pane of the project browser. The top source record is a comment. For more information on the icons in Apple Dylan, see the section "Using icons in Apple Dylan" on page 88.

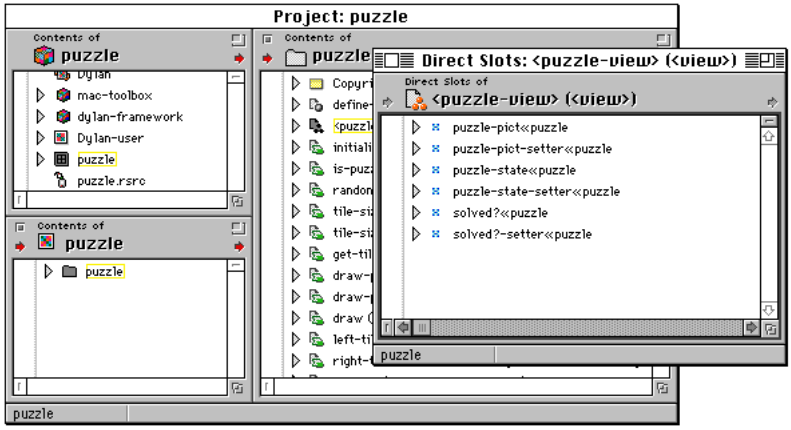
   In the following figure you can see all the direct methods for <puzzle-view> listed in the Direct Methods browser that opens. You can investigate each of these source records further, if you wish, expanding them to see their source code or opening other browsers for them. If you edit their source code here, the changes are updated in all the browsers immediately. To save the code changes, click in any pane where the source code is displayed (even if it is

currently out of view due to scrolling) and use the Save command from the File menu.

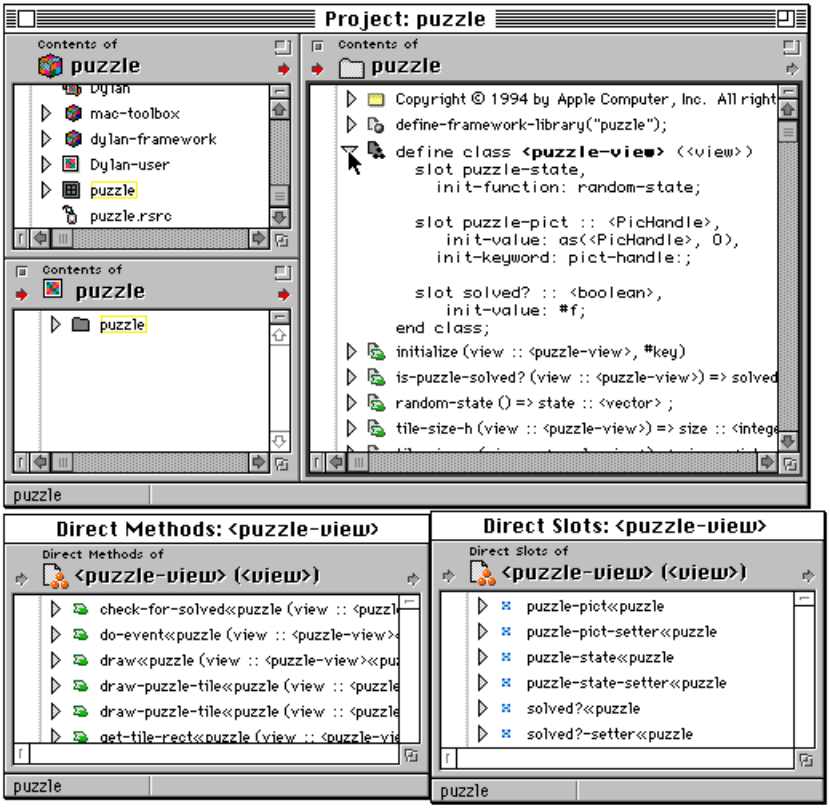


Drag the Direct Methods browser off the project browser before continuing.

3. **Select the class <puzzle-view> in the project browser and choose Direct Slots from the Browse menu.**
   Another browser, Direct Slots, opens with <puzzle-view> as its basis as well. This browser lists all the direct slots for <puzzle-view>. You are now able to view two aspects of <puzzle-view> in the new browsers, its direct methods and its direct slots.

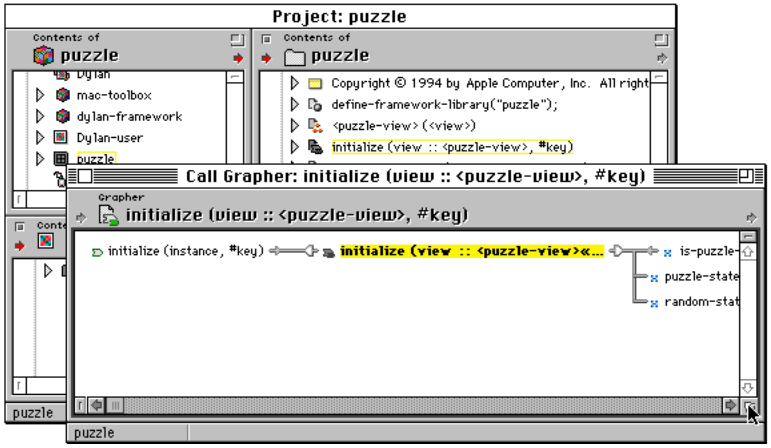


If you drag the Direct Slots browser off the project browser and expand the <puzzle-view> source record inline, you see its source code inline. As shown in the following figure, you would be viewing three aspects of <puzzle-view> simultaneously. In fact, a built-in browser named Info for Selected Class is very similar to this. For more information on it, see the task "Using the browser Info for Selected Class" on page 53.



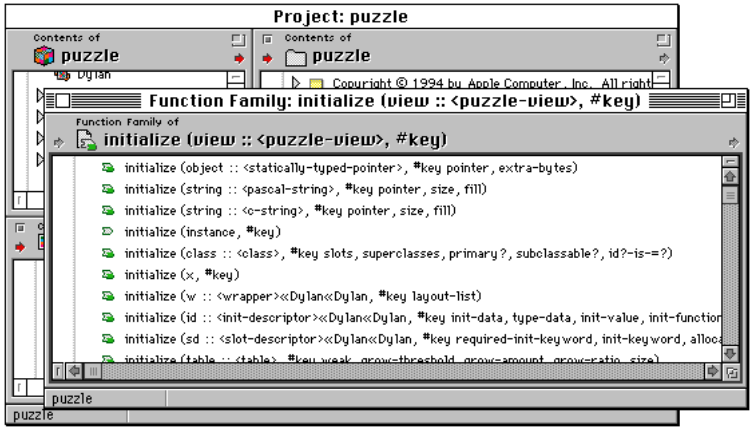
4. **Select a source record for a method in the right-hand pane, such as initialize, and choose Call Grapher from the Browse menu.**
   The initialize method is the third source record from the top in the right-hand pane of the project browser.

   In the following figure you can see all the callers and callees of initialize graphed in the Call Grapher browser that opens. You can investigate these calls further, if you wish, by expanding or collapsing the arrows on the bars that graph the calls. For more information on how to use grapher panes, see the task "Using the browser Info for Selected Class" on page 53.



Drag the Call Grapher browser off the project browser before continuing.

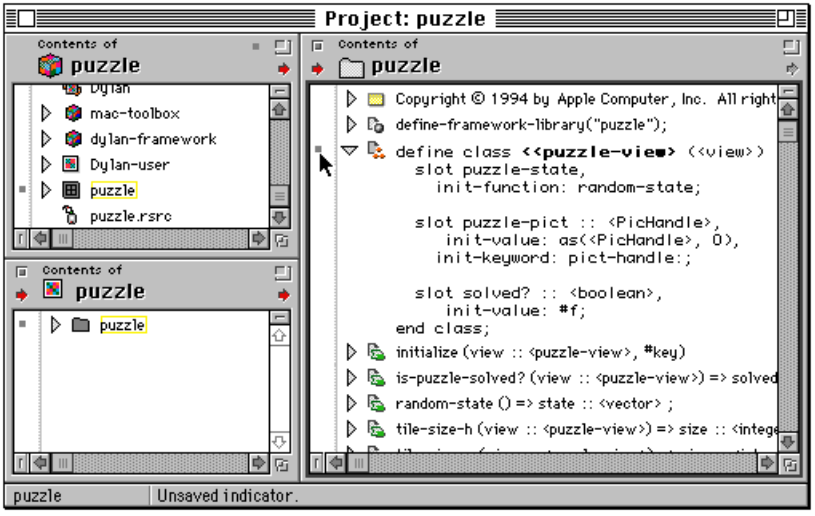
5. **Select initialize in the project browser and choose Function Family from the Browse menu.**
   Notice that another browser, Function Family, opens with initialize as its basis as well. This browser lists all the other initialize functions in the active project. You are now able to view two aspects of initialize in the new browsers.



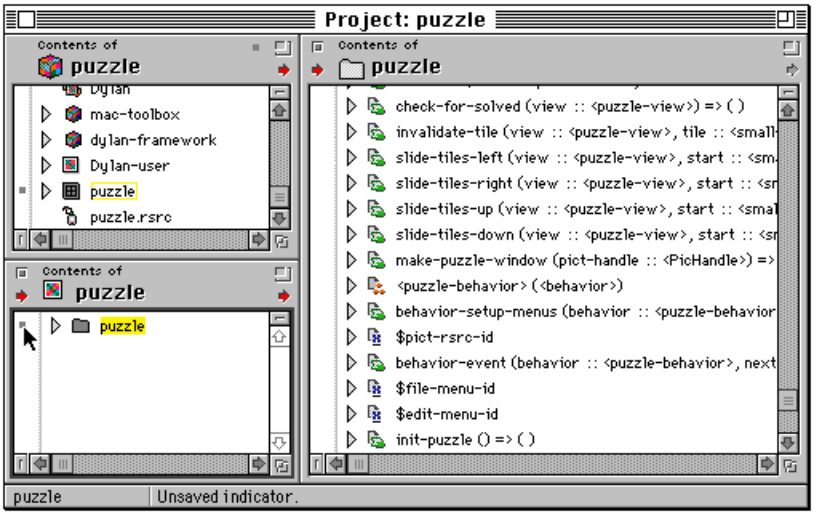
Drag all the browsers off the project browser before continuing.

6. **Expand the class <puzzle-view> in the project browser and type an extra < into its definition.**
   Notice that a small square appears in the gray, status indicator column to the left of the source record. This is a status indicator. If you click the status indicator, its name appears on the information line at the bottom of the browser. In this case, it is the Unsaved indicator, meaning that you have changed this source record but haven't saved it yet.

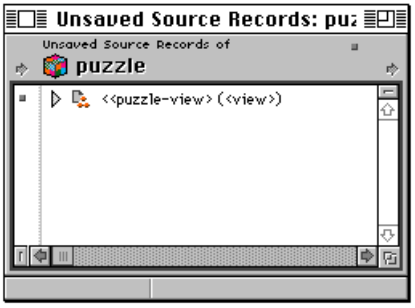
   There are several other status indicators in Apple Dylan, including the Uncompiled indicator and Warnings indicator. You can see the list of indicators using the Status Indicators command on the Browse menu. For more information on using them, see the chapter "Using Apple Dylan" on page 95.



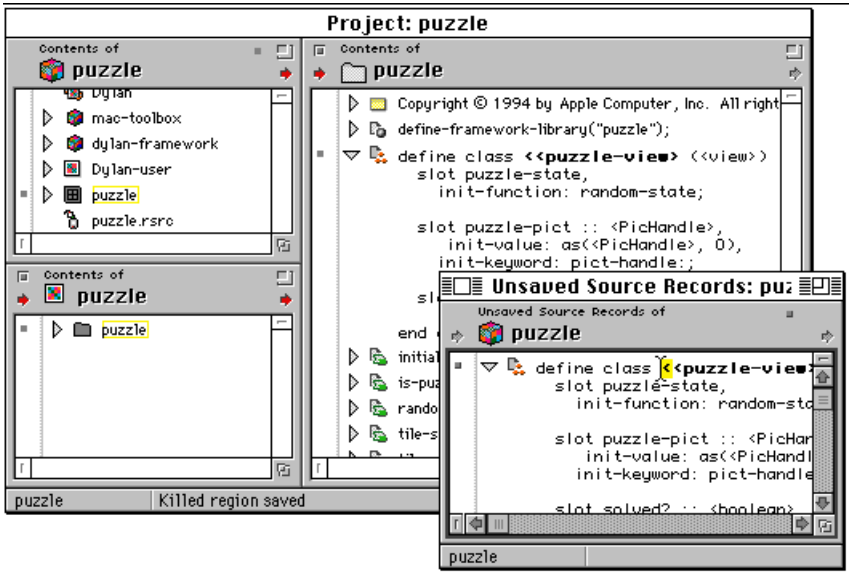
7. **Scroll down the source records in the right-hand pane and expand some source records until <puzzle-view> is no longer visible.**
   Notice that, although you cannot see the unsaved source record anymore, you can see that the source folder it's in, puzzle, has an Unsaved indicator in the lower-left pane marking it as containing something unsaved. The puzzle module in the root pane has one as well. If you had gone on to editing other source records and forgotten exactly which source records were still unsaved, you might have a difficult time finding them. In that case, you would want a way to list all the unsaved source records in the project.



8. **Choose Unsaved Source Records from the Browse menu.**
   The Unsaved Source Records browser opens, listing <puzzle-view>. Notice that the puzzle project icon appears in the pane's header of the Unsaved Source Records browser. It does not matter if you have anything selected when you choose this command; as with all browsers in the third section of the Browse menu, they work on the entire active project regardless of selection.

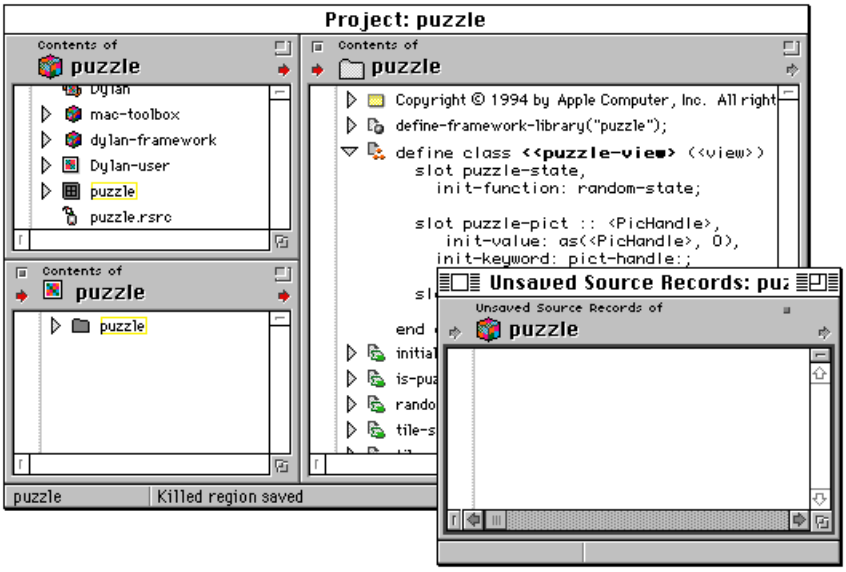


9. **Expand the source record and delete the extra < from the source code.**



You will notice that the code is still marked as unsaved with a status indicator and still listed in the Unsaved Source Records browser.

10. **Click in the Unsaved Source Records browser and choose Save from the File menu.**
Notice that <puzzle-view> is no longer listed in the Unsaved Source Records browser.



Close the empty browser or move it off the project browser before continuing. Notice that the source code in the project browser has been updated, as has all the status indicators.

Linking panes and browsers

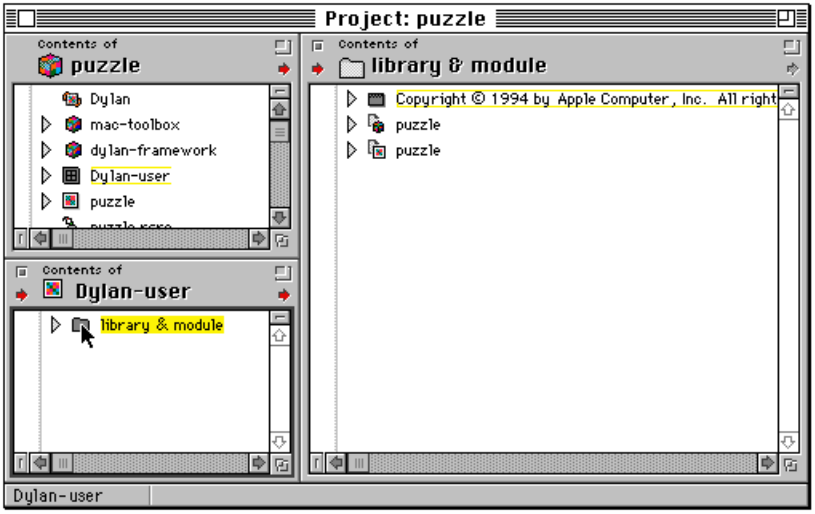
The panes in a browser can work together through links. The contents of an object you select in one pane show up in another pane through a link between them. You can create, remove or redirect links.

To see a link's origin or destination, hold the mouse down over the link's **inbox** or **outbox**. The inbox is a small right-pointing arrow at the left edge of a pane's header. The outbox is a small right-pointing arrow at the right edge of a pane's header. When you hold down the mouse on an inbox or outbox, the name of the pane it's linked to is displayed. If no link has been established, no name is displayed and the inbox or outbox is gray. When a link has been created, the inbox and outbox turn red. You can create a link by dragging the outbox from one pane onto the inbox of another pane. An aspect of the object selected in the first pane is displayed in the second pane. That object selected in the first pane is called the **basis** of the second pane.

You can open one of the browsers on the Browse menu or you can create a new browser of your own, either by double-clicking on an object or selecting it and choosing the New Browser. A new browser has only one pane, so if you want to have more than one pane in the browser, you split the pane in two. The new pane is automatically linked to the original pane. You can close an individual pane within a multi-pane browser by using its close box in the upper left corner of its header.

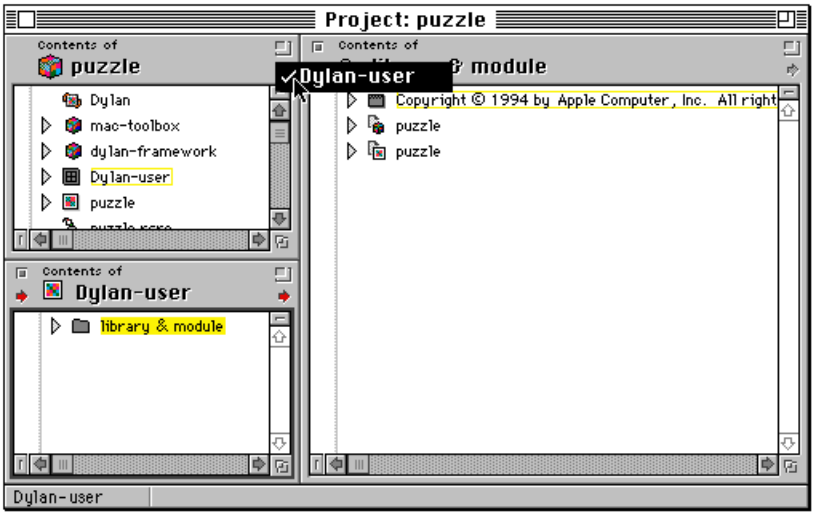
1. **Open the puzzle project, if it's not open already, and make it the active project.**
   The sample projects can be found in the Sample Code folder, or by using the alias in that folder.

   The following figure shows the project browser for puzzle with the module Dylan-user selected in its root pane and the source folder library & module selected in its lower-left pane. The contents of the source folder are displayed in the pane on the right.



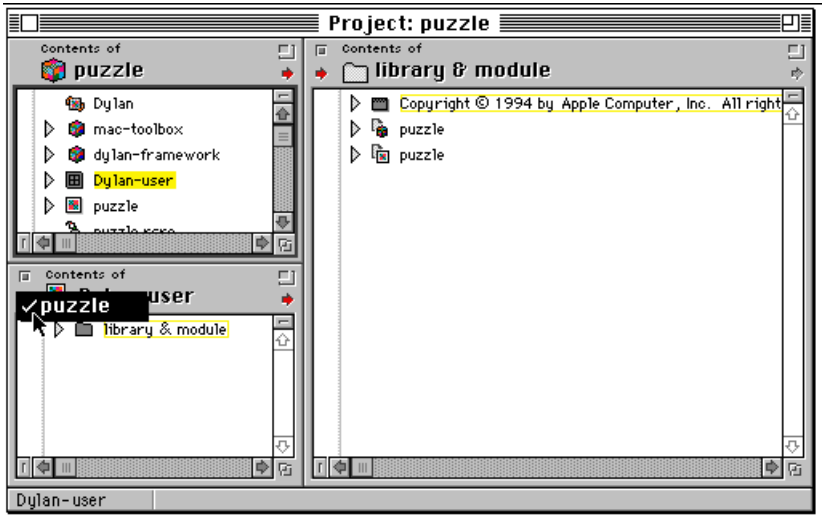
2. **Hold down the mouse on the outbox of the root pane without releasing the mouse.**
   The outbox displays the outgoing link, which in the following figure is from the puzzle project's root pane to the lower-left pane.

   Release the mouse when you have seen the link's name.



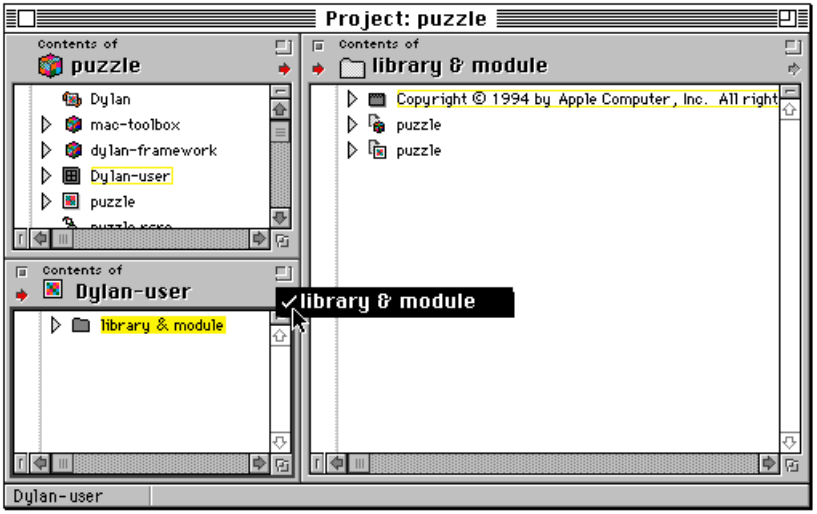
3. **Hold down the mouse on the inbox of the lower-left pane.**
   The inbox displays the incoming link, which in the following figure is from the puzzle project's root pane above.

   Release the mouse when you have seen the link's name.



4. **Hold down the mouse on the outbox of the lower left pane.**
   The following figure shows the destination of the outbox's link is the pane to the right, which has "library & module" as its basis, as noted in its pane's header.

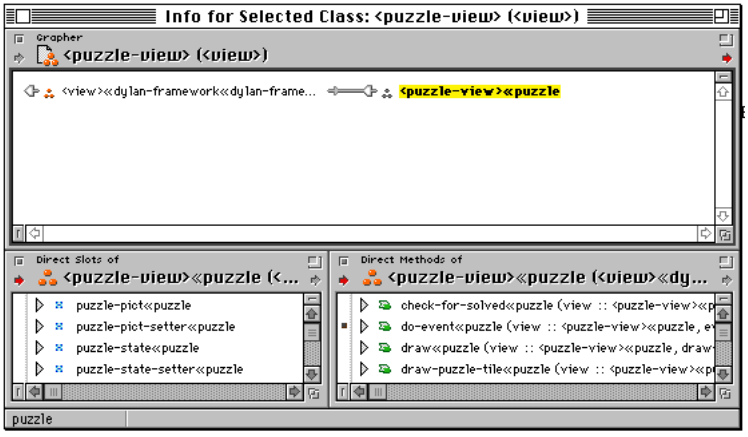
   Release the mouse when you have seen the link's name.



5. **Select the puzzle module in the root pane and the puzzle source folder in the lower-left pane.**

6. **Select a class, such as <puzzle-view> in the right-hand pane and choose the browser Info for Selected Class from the Browse menu.**
   Leave the project browser open as you work with the other browser.

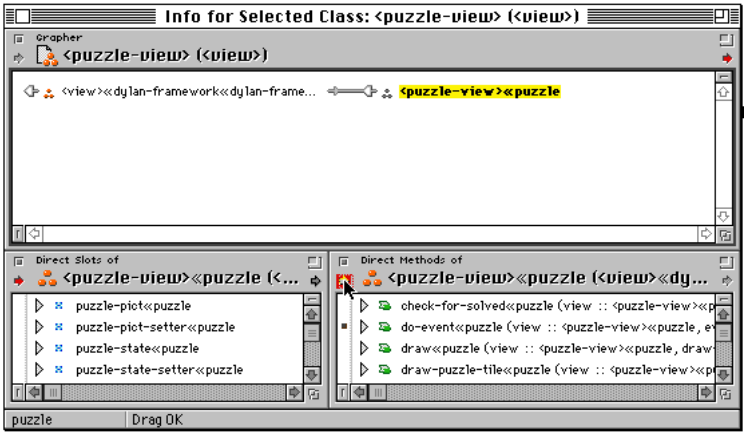
   The browser shown in the following figure opens with <puzzle-view> as its basis, as noted in the browser's name and in its root pane's header. The class <puzzle-view> is automatically selected in the grapher pane at the top of the browser, while its direct slots and direct methods are listed in the two lower panes, as shown in the following figure.



If this browser were already open, you could have dragged the <puzzle-view> class from the project browser to the inbox of the root pane in this browser. This would have made <puzzle-view> the basis of this browser. Dragging an object from one browser to the inbox of the root pane on a separate browser is similar to linking them, but it does not create a link. This is drag and drop behavior that is common to all browsers. The dragged object is simply the basis for the separate browser; the two browsers are not linked. The project browser's outbox and the separate browser's inbox are still gray and no link exists.

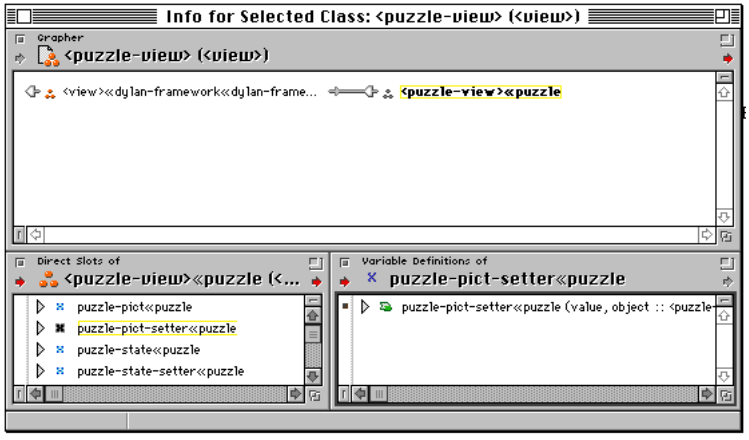
7. **Examine all the links on the inboxes and outboxes for the Info for Selected Class browser, just as you did for the project browser.**
   Notice the inboxes and outboxes. The two outgoing links in the lower panes are not red; they are not linked to anything yet.

8. **Drag the outbox for the lower-left pane to the inbox of the lower-right pane.**
   In the following figure the inbox for the lower-right pane is red as the outbox from the other pane is being dragged onto it.

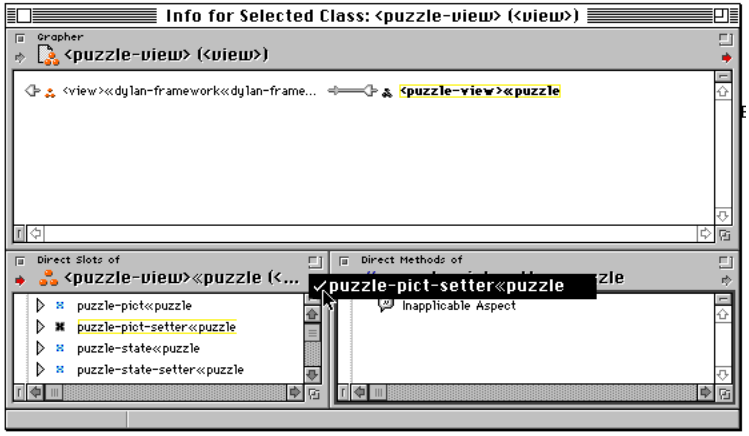


Release the mouse on the inbox and the two panes are linked.

9. **Click the puzzle-pict-setter source record in the lower-left pane.**
Notice that the selected source record is now the basis for the lower-right pane. In the following figure the source record puzzle-pict-setter is selected in the lower-left pane and is named as the basis of the lower-right pane in its header. Examine all the links.



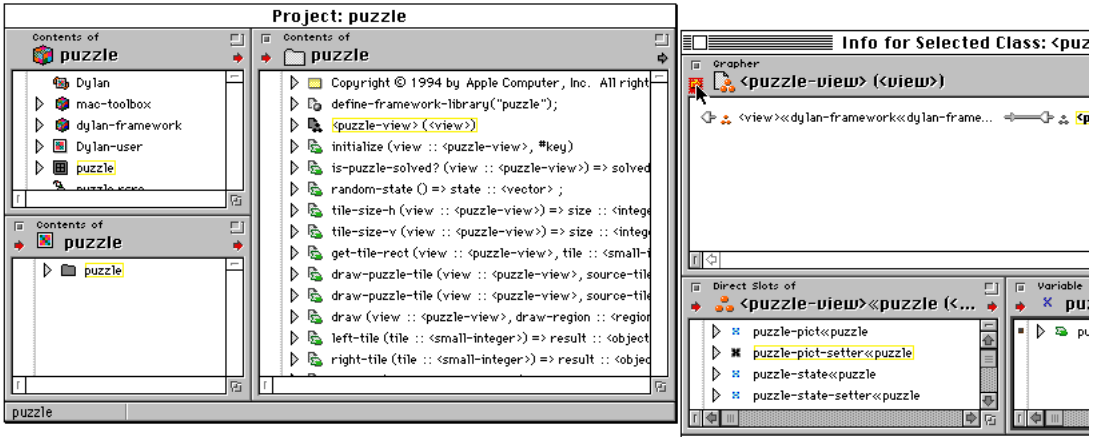
Notice that the outbox of the lower-left pane names the lower-right pane as its destination, as shown in the following figure.



Direct methods is not a valid aspect for puzzle-pict- setter, but you can easily change it to show one that is, such as the variable definitions of it in the active project. For more information on how to change aspects in Apple Dylan, see the task “Changing aspects” on page 46.

10. **Drag the outbox from the right-hand pane on the project browser to the inbox of the root pane on the Info for Selected Class browser.**
You can link two separate browsers by dragging the outbox from a pane in one browser onto the inbox of any pane in the other browser. This allows you to create complex sets of browsers.

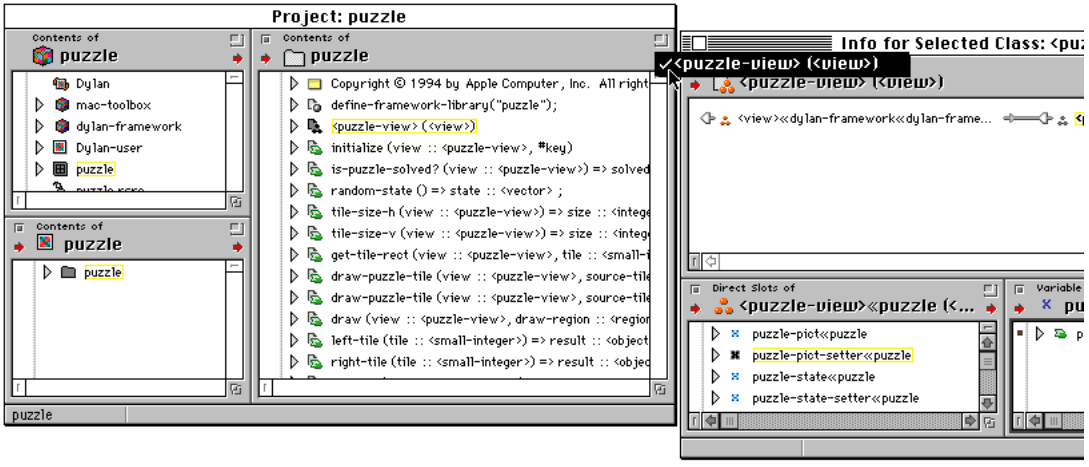
    Notice that the inbox of the Info for Selected Class browser turns red as you drag the outbox from the project browser onto it, as shown in the following figure.



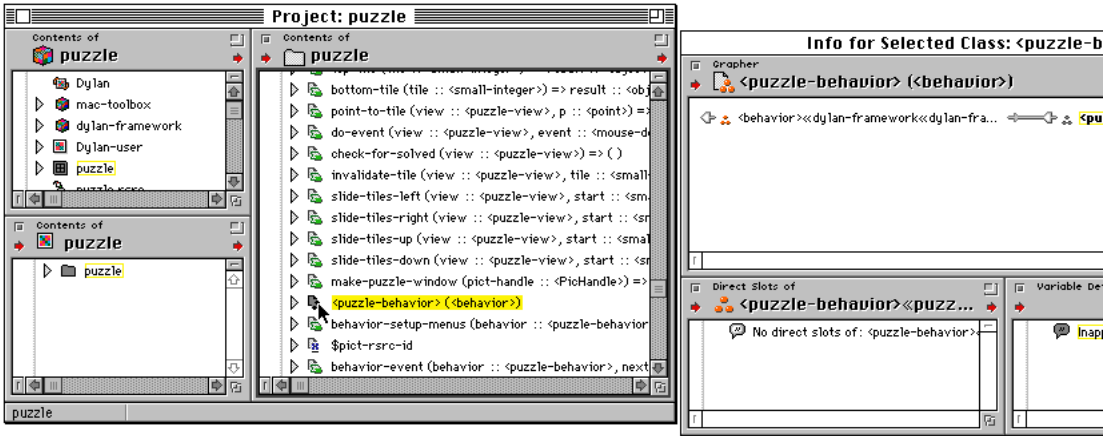
11. **Release the mouse.**

    You will notice that the outbox and inbox of the two browsers are now red. The outbox on the project browser names the <puzzle-view> class as its basis and the <puzzle-view> class is also the basis of the Info for Selected Class browser. Any other class you select in the right-hand pane of the project browser will be displayed in this Info for Selected Class browser.

    This link applies only to this particular Info for Selected Class browser. Furthermore, if you close it, this link is lost.



12. **Click the class <puzzle-behavior> in the project browser to see it as the new basis for the Info for Selected Class browser.**
    In the following figure, <puzzle-behavior> has been selected in the project browser, which causes it to become the basis for the other browser. Examine the outgoing link from the project browser.



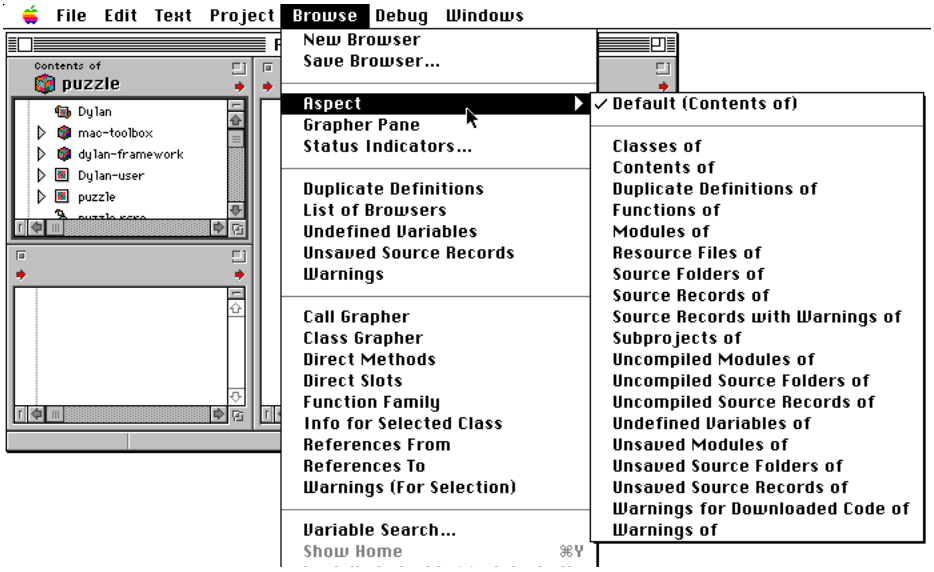
Showing different aspects of objects

You can view each object in Apple Dylan in different ways. The various aspects available for objects include some basic qualities, such as a source folder's contents, or relationships between objects, such as what calls what, class relationships, families of functions, etc. Aspects are only available for the active project.

An aspect is displayed in a pane. You can see the list of aspects available by clicking in a pane and choosing the Aspect command from the Browse menu. A list of available aspects is displayed. If you do not select any object in the pane when you click in it, the aspects listed are for the basis of the pane. If you select an object in the pane, the selected object's aspects are listed instead.

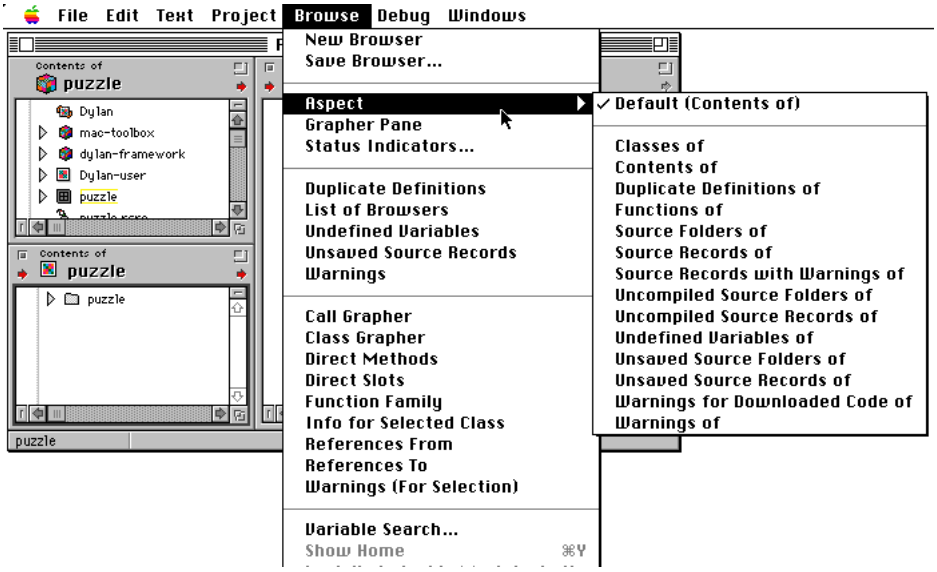
1. **Open the puzzle sample project, and make sure it's the active project.**
   The sample projects can be found in the Sample Code folder or using the alias in it.

2. **Click in the puzzle project's root pane, without selecting anything, and then choose Aspect from the Browse menu.**
   On the submenu of the Aspect command, the aspects available for a project are listed. All projects have the same aspects available for them. You could choose to view an aspect other than Contents of this project, such as Classes of, but if you do, choose Default (Contents of) again before continuing.



3. **Select the puzzle module in the project's root pane.**
   Notice that the pane displaying the contents of the module (the pane on the lower left) has the aspect "Contents of" in its header. This is the default aspect for a module. The puzzle module's name and icon appear in the header of the pane at the bottom as well.

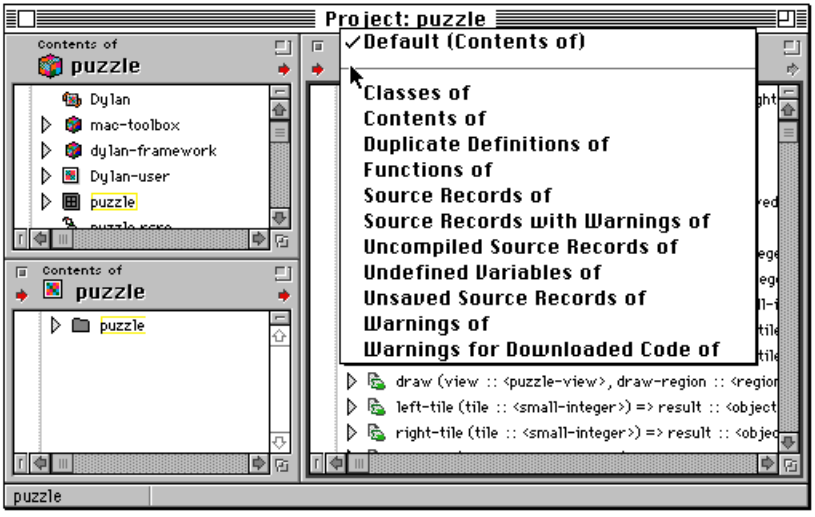
4. **Click in the lower-left pane without selecting anything, and then choose Aspect from the Browse menu.**
   In the following example, notice that the top line of the list of aspects is the default aspect for modules, Contents of. It is the contents of the puzzle module that you see in the bottom pane. You can select an aspect from the list to view a different aspect. For our purposes, continue on without choosing a different aspect or, if you do, reselect the default before continuing.



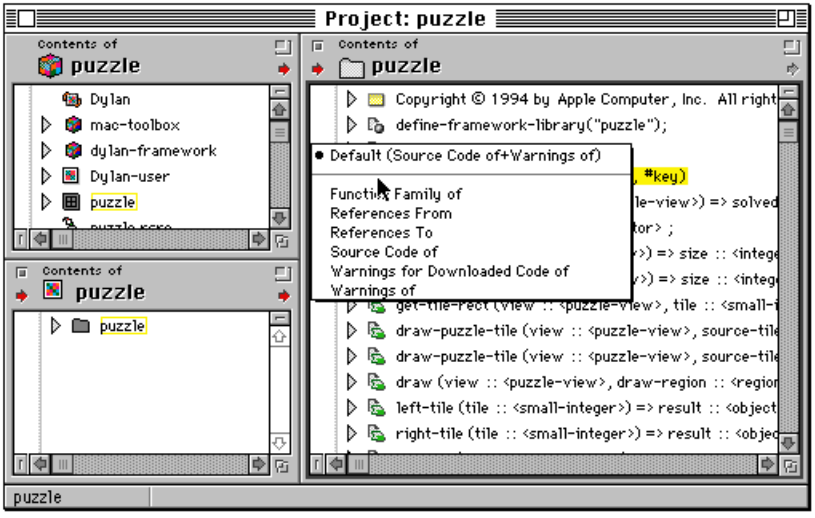
5. **Select the puzzle source folder in the lower-left pane and then click in the right-hand pane without selecting anything.**
   The right-hand pane now displays the contents of the puzzle source folder.

6. **Display the pop-up list of aspects from the header of the right-hand pane.**
   You can display an object's list of aspects from the header of the pane its contents are displayed in. You do this by clicking in the pane without selecting anything, holding down command-option, and then holding the mouse down on the object's name or icon in the header. You will see all the aspects available just as you would if you had used the Aspect command from the Browse menu.

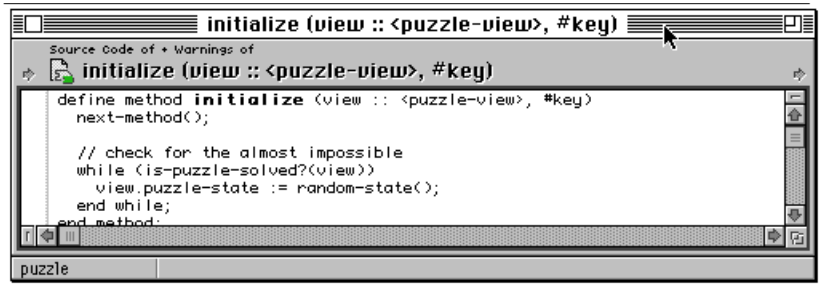
   In the following example, the aspects available for source folders are displayed on the pop-up list because the puzzle source folder is the basis for the pane. For our purposes, don't select one or, if you do, reselect the default before continuing.



7. **Hold down the Command and Option keys, then click the initialize source record in the right-hand pane.**
   The list of aspects for source records is displayed. Notice that its default aspect is Source Code of +Warnings of.



8. **Double-click on the source record initialize.**
   If you double-click on an object to open a new browser for it, the default aspect is displayed. The new browser that opens has the object as its basis, so the object's icon, its name, and the name of its default aspect appear in the header of the new browser's pane. In the following example, the default aspect Source Code of +Warnings of is named in the header.



Changing aspects

The easiest way to see a different aspect for an object is to select the object and choose a built-in browser from the Browse menu. Several browsers have been built into the development environment for your convenience. They each show a different aspect, as noted in their names on the Browse menu. When you choose one, a new browser opens displaying the aspect for the selected object.

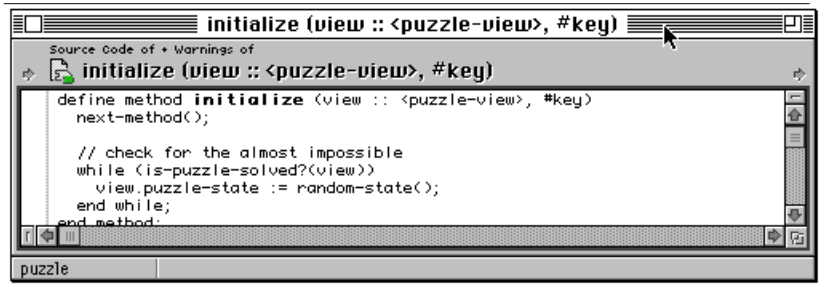
However, not all aspects have built-in browsers already made for them. Also, you might want to change the aspect in a pane you created in a custom browser or some other existing pane, not open an entirely separate browser. In that case, you can choose to change the aspect within a pane. Also, if you change the basis for a pane, the new basis might have a different default aspect, so that new aspect will consequently be displayed in the pane.

Many aspects are available only for the active project.

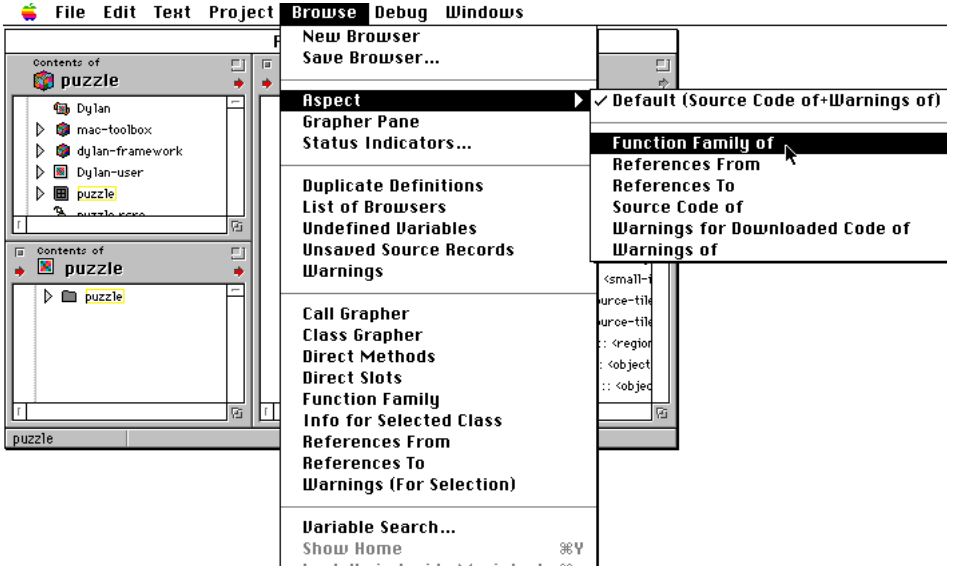
If you change the aspects in a browser and want to keep that browser configuration for use later, you must save it before closing the browser. See the task "Saving a browser configuration" on page 61 for more information.

1. **Open the puzzle sample project and make sure it's the active project.**
   The sample projects can be found in the Sample Code folder or using the alias in it.

2. **Double-click on a source record, such as initialize.**

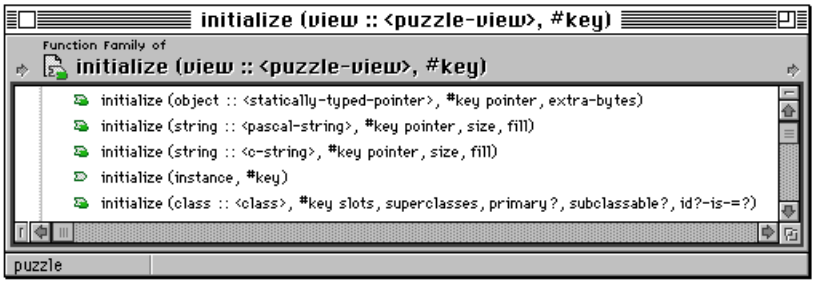


3. **Click in the pane but don't select anything, then choose the Aspect command from the Browse menu.**
   You will see that the aspects for a source record are listed. Its default, Source Code of +Warnings of, is the first item on the list. You can choose to see any of the other aspects on the list and can go back to the default by choosing it again later.



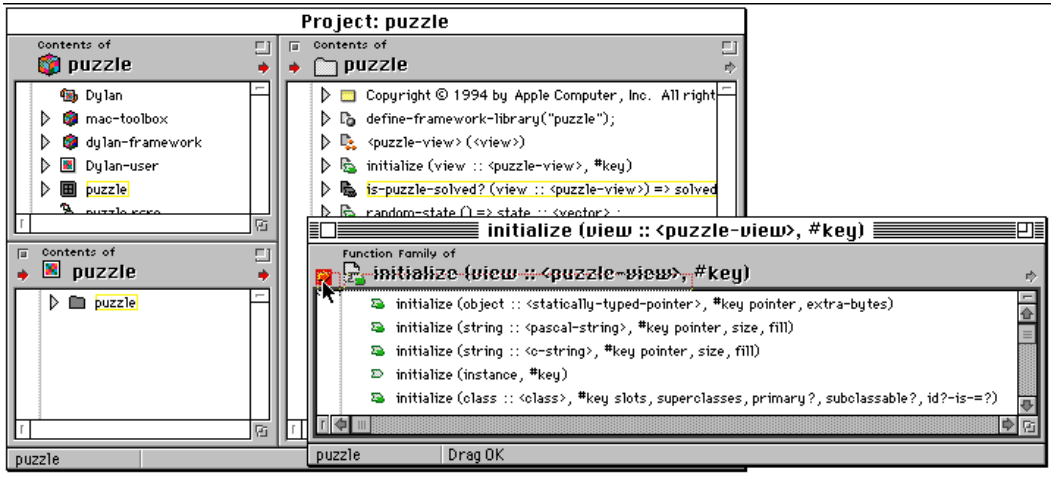
4. **Select another aspect from the list, such as Function Family.**
   You will see the aspect appear in the pane, replacing the previous aspect. No new pane or browser opens. Notice that the aspect named in the header of

the pane has changed in the following figure. The pane's basis, initialize, has not changed, but its aspect has.



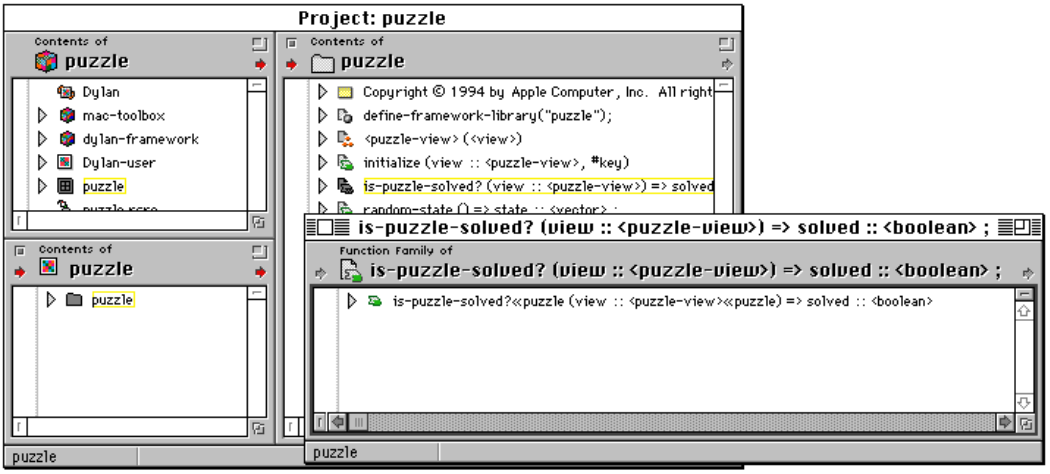
5. **Select an object, such as the source record is-puzzle-solved in the project browser.**

6. **Drag the is-puzzle-solved source record to the inbox on the pane's header in the new browser.**
   The inbox, a small arrow on the left of the pane's header, turns red when the object is on it.



7. **Release the mouse with the cursor still over the inbox.**
   The is-puzzle-solved source record becomes the new basis for the pane in the Function Family browser. However, the inbox to the Function Family browser is not red. This is because you have not created a link by dragging

in another source record, you have only changed the basis of the pane. The icon and name of is-puzzle-solved are shown in the pane's header in that browser.

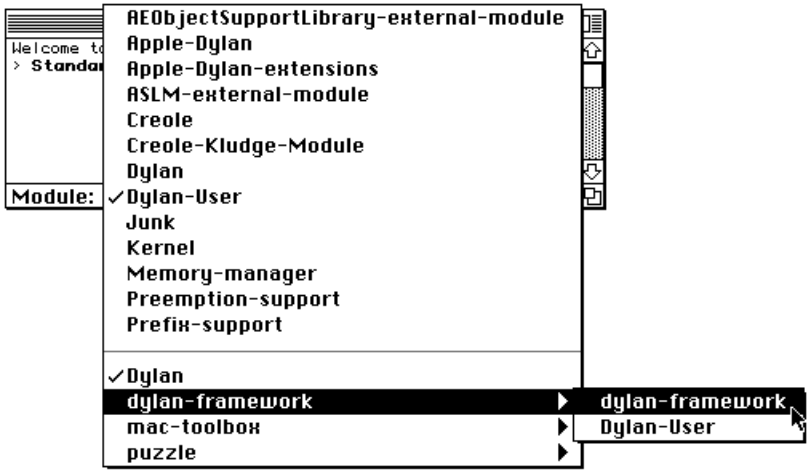


Ordinarily when you expand an object within a pane, you see its default aspect. For example, if you expand a source record its Source Code of +Warnings of aspect is displayed inline. You can change that aspect to another aspect by selecting the object's name or icon and then using the Aspect command to select a different one. The new aspect is revealed inline with the object; no new pane is created, nor is a new browser opened.

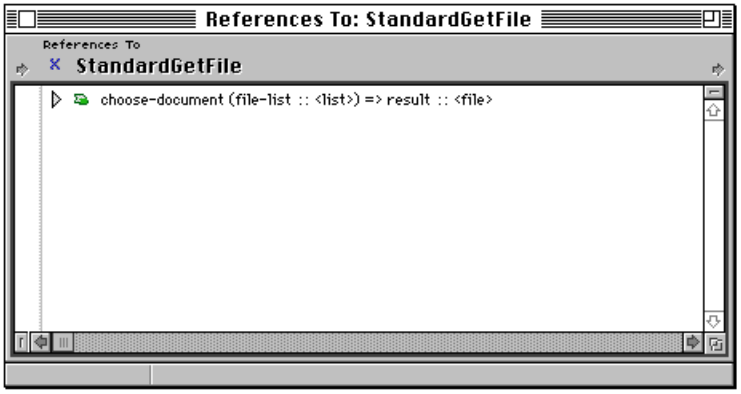
Using the browser References To

You can use browsers to gather information when writing code. The References To browser displays all the references to a selected class, function or variable in a project. You can use it to see such things as what makes references to a certain slot or to use the sample projects and the framework for useful code. For example, if you wanted to create an Import Picture command in your application, you might assume that StandardGetFile or some variant was probably what would help you and that it's probably in the framework. The active project in the following example of this usage is puzzle.

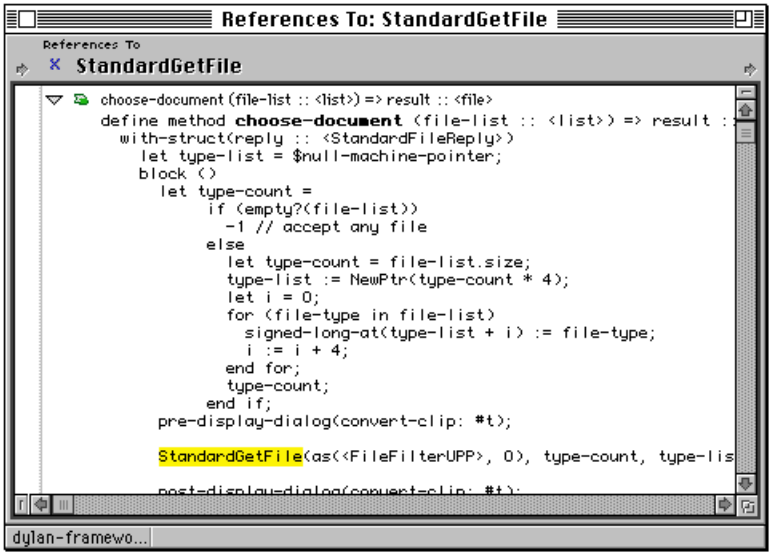
1. **Open the puzzle sample project and make sure it's the active project.**
   The sample projects can be found in the Sample Code folder or by using the alias in it.
2. **Type the name of the class, method or variable in question, StandardGetFile in this case, into the Listener.**
   This simply identifies the class or method you want to investigate. You could type it into a source record in a pane and select it there.
3. **Choose the dylan-framework module from the Listener's Module popup.**
   You must set the module you want to search in using the Module popup in the lower-left corner of the Listener window. For more information on using the Listener, see the section "Running an application in Apple Dylan" on page 144.



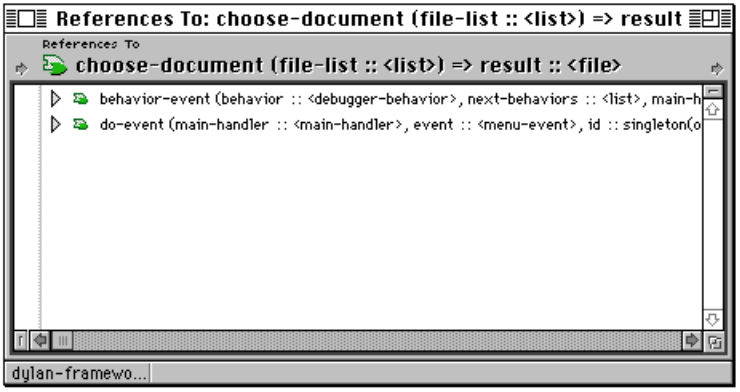
4. **Select StandardGetFile in the Listener and choose the References To browser from the Browse menu.**
   The References To browser opens and shows that StandardGetFile is called by a framework function called choose-document.



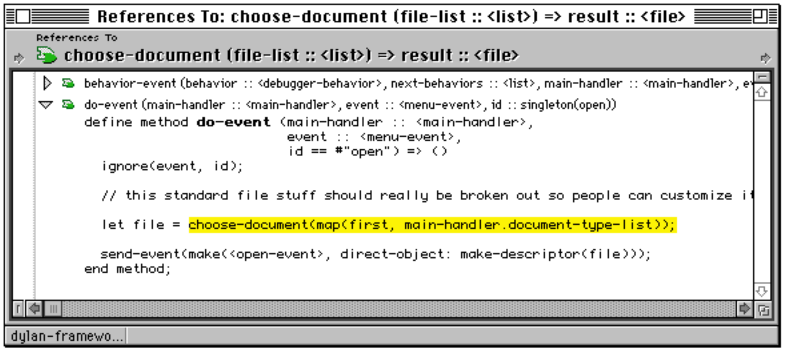
5. **Expand the source record for choose-document and search for StandardGetFile.**
   StandardGetFile is not highlighted automatically. You have to search for it. The arguments to StandardGetFile are displayed in the following example. You might conclude that it is a bit too raw for your purposes and that you would be better off using the calling function choose-document since the framework has provided a wrapper.



6. **Select choose-document and use the Browse menu to select the References To browser again.**
   We want to see how choose-document is used. We see there is a behavior-event and a do-event.



7. **Click the disclosure triangle for do-event.**
   Choose-document and its arguments are displayed so you can see how to make a call to it.



Using the browser Info for Selected Class

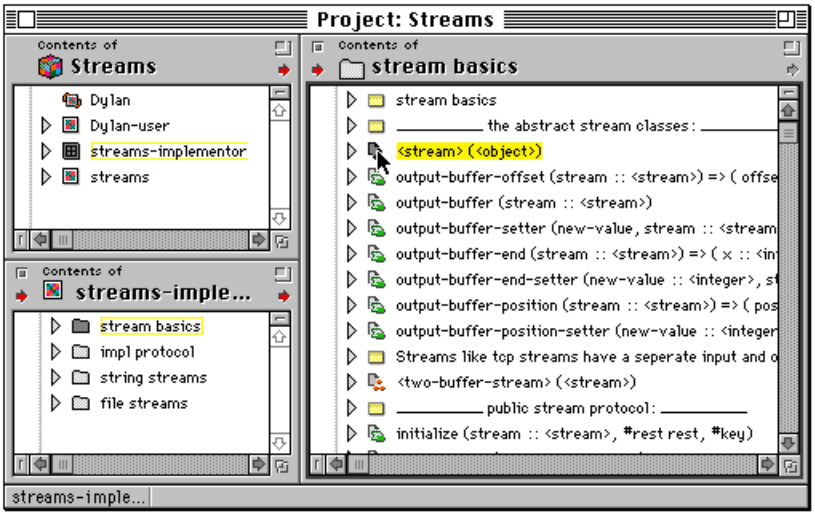
You use the Info for Selected Class browser to see several aspects of a class. The top pane is a graph of the class hierarchy for a class and the two other panes in this browser show a list of its direct methods and a list of its direct slots.

This browser's root pane is a grapher pane, which means it visually represents the relationships between the objects on a graph instead of listing objects. Click the arrows on the graph to expand and collapse it. Other built-in browsers with grapher panes are the Call Grapher and Class Grapher browsers. You can have several of each of these browsers open at once. See also the Grapher Pane command on page 213 for more information.

Only the basis of the browser is fully represented on the graph in a grapher pane; you have to change the basis of the browser to get full representation for another object. For classes, that means you can see multiple inheritance and subclass relationships only for the basis of the browser.

As with other browsers, when you select an object before opening the browser, you make that object the basis for the browser. Then, to change the browser's basis, you can drag any other appropriate object, such as a class, into its inbox from any browser. However, in grapher panes you can also change the basis of the browser by doing a control-click on another object within the browser to make that object the new basis.

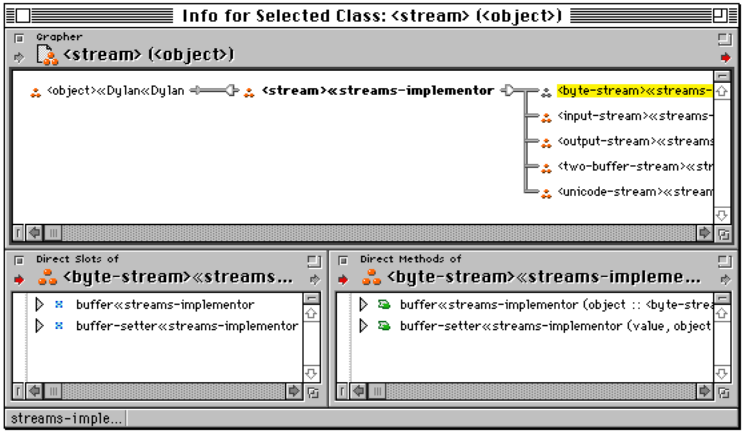
1. **Open the Streams sample project and make sure it's the active project.** The sample projects can be found in the Sample Code folder or by using the alias in it. If you had another project open before opening Streams, you need to click in the Streams project once you open it, and then use the Activate Project command from the Project menu. If Streams were not the active project, the project browser's name would read "Project: Streams (inactive)" and the root pane's name would read "Streams (inactive)".
2. **Select a class in the Streams project, such as <stream> in the stream basics source folder.**



3. **Choose the browser Info for Selected Class from the Browse menu.**
   The browser opens, showing the graph for <stream> along with its direct slots and direct methods.

   The graph displays the five subclasses of <stream>. Notice that the class <object> has no arrow to its left. That means it is the root of this hierarchy.

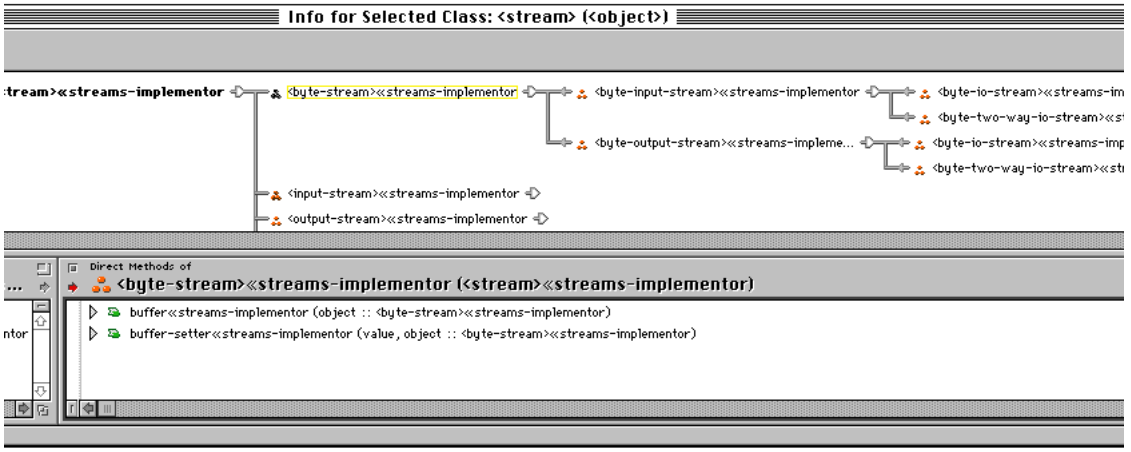
4. **Click the first subclass on the list, <byte-stream>.**
   Its direct slots and direct methods are listed in the two lower panes, as shown in the following figure.



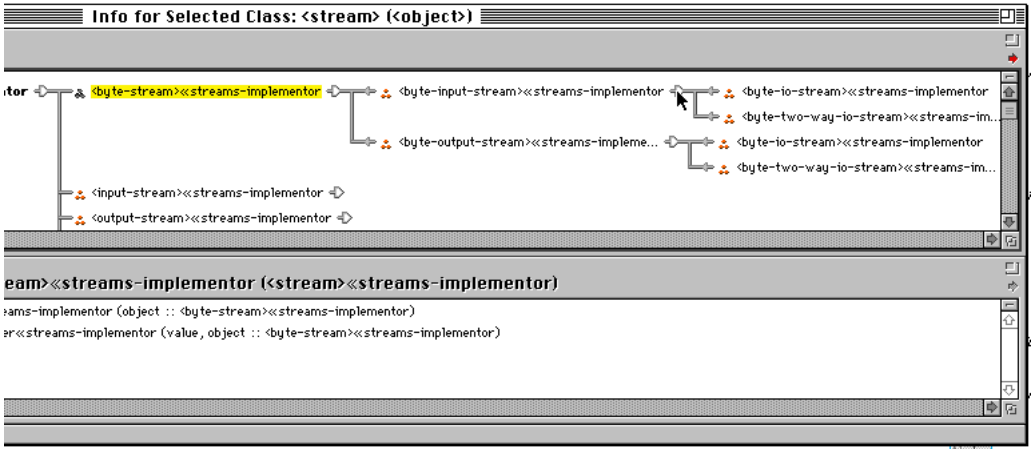
Continue clicking the other subclasses, to see their direct methods and direct slots, if you want. Resize the browser slightly so you can see the full names of the five subclasses.

5. **Click the arrow to the right of <byte-stream> on the graph.**
   This further expands the graph. Notice that there are two more subclasses, but they are not fully visible. Resize the browser more to see the full names of the two subclasses.

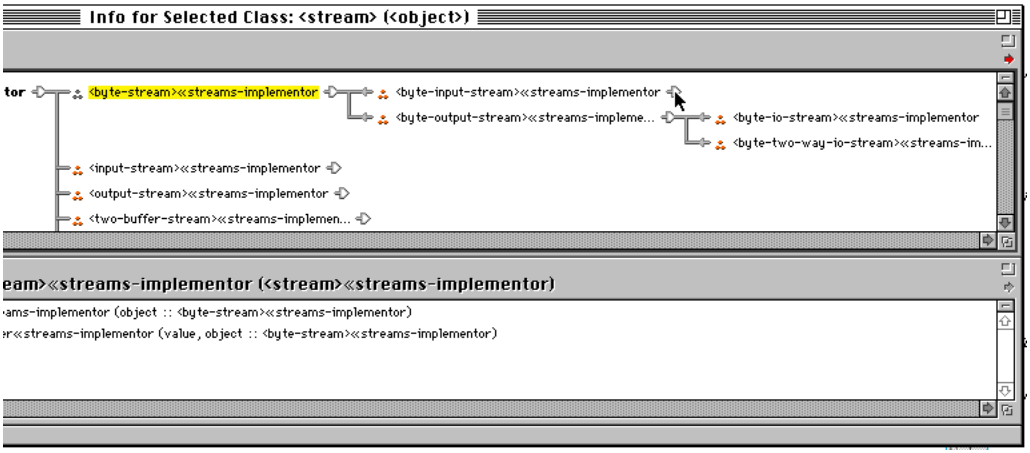
   Continue expanding subclasses until there are no more arrows to the right of any subclass, if you have room on your monitor. You could reveal all the classes of the hierarchy by continually expanding and resizing the browser, but you would need a very large monitor.



6. **Select an arrow on a bar separating two classes and click the arrow to collapse that part of the hierarchy.**
   The following figure shows the arrow to click to collapse the graph at that point.

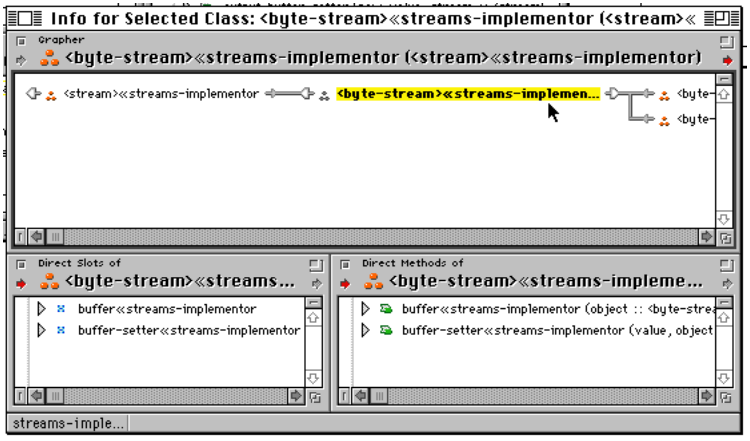


The following figure shows the result of clicking the arrow.

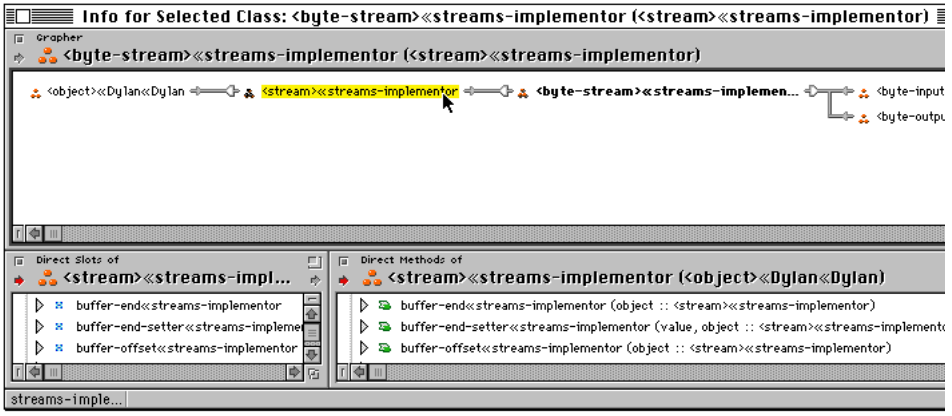


Collapse the hierarchy back to its original size.

7. **Control-click on a class on the graph, such as <byte-stream>.**
   This changes the basis of the browser to the <byte-stream> class. The following figure shows the result of making <byte-stream> the new basis for the browser. The <byte-stream> class has also been selected on the graph to show its direct slots and direct methods. You will now be able to examine multiple inheritance and the subclasses of <byte-stream> in depth.



8. **Click the <stream> class in the grapher pane.**
   Notice that you can no longer see all the subclasses of <streams> as you could when it was the basis of this browser.



Customizing browsers

You can customize any browser, either an existing one or a new one you create. You create a new browser by double-clicking on an object in a pane, which will open a new, separate browser with that object as its basis. You can also create a new browser by selecting an object and using the New Browser command on the Browse menu.

To customize a browser, you can change the links between its panes as described previously in this section. Furthermore, you can change the aspects visible in the panes. For more information on aspects, see the tasks “Showing different aspects of objects” on page 42 and “Changing aspects” on page 46.

You can also add panes or resize existing panes. A new browser has one pane and its basis is the object selected prior to creating it. The selected object is specifically the basis for the root pane of the new browser and can, therefore, be considered the basis of the new browser. The default aspect for the selected object dictates the type of aspect displayed in the new browser’s pane.

If you want more than one pane in the new browser, you can split the pane in two. Each of the two new panes can also be split, both horizontally and vertically. You split a pane with its splitter controls; the horizontal splitter

control is a short, horizontal bar just above the scroll bar on the right of a pane. The vertical splitter control is a short, vertical bar just to the left of the scroll bar at the bottom on a pane. To split a pane, hold down the mouse on the splitter control and then drag the dashed line that appears to wherever you want the pane split. Another way to split a pane is to drag an outbox onto a splitter; the pane splits in half and a link is created from the original pane to the new pane.

You can resize existing panes to create different relative sizes. The resize control in the lower-right corner of each pane and each browser can be dragged to create new sizes. You can also drag a horizontal line up and down and a vertical line from side to side between panes.

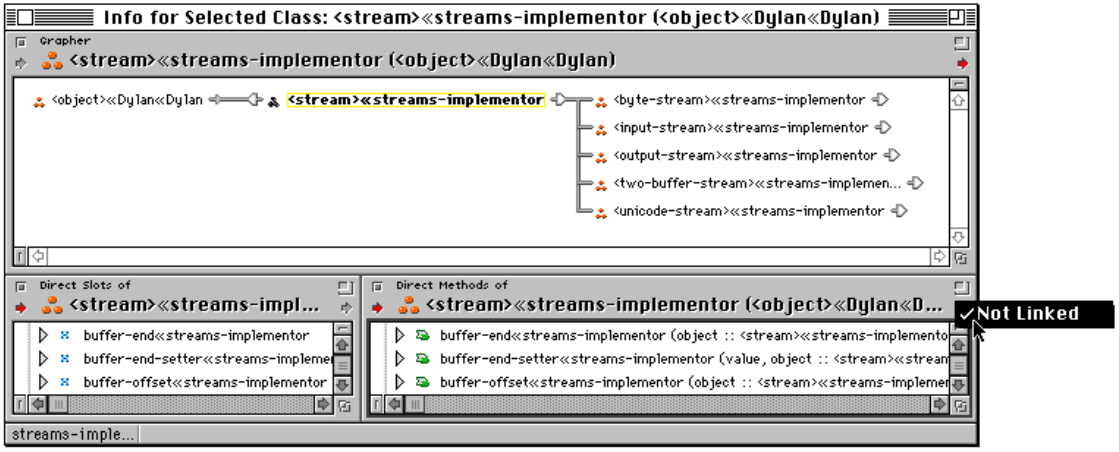
You can close a pane by clicking its close box in the upper left corner of its header. If you want to save the configuration of a customized browser, do so before you close it. For more information on saving browsers, see the task "Saving a browser configuration" on page 61.

In the following task, the browser Info for Selected Class is customized by adding a pane.

1. **Open the sample project Streams, if it's not already open, and make sure it's the active project.**
   The sample projects can be found in the Sample Code folder or by using the alias in it.

2. **Select the <stream> class in the stream basics source folder, and choose Info for Selected Class from the Browse menu.**
   The Info for Selected Class browser opens with <stream> as its basis.

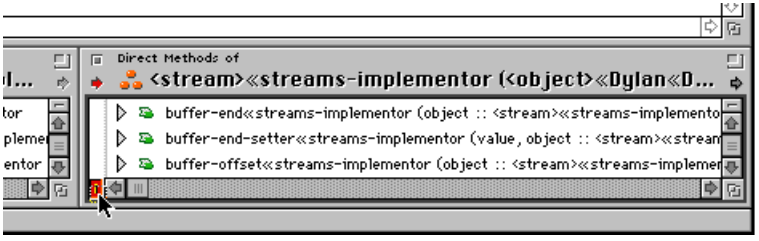
3. **Examine the outbox of the lower-right pane.**
   Notice that it is not linked to anything, so the outbox in the pane's header is gray. In addition, the outbox says "Not Linked", as shown in the following figure.



4. **Drag the outbox of the pane onto its own vertical splitter control.**
   You can split a pane in one of two ways, by dragging its outbox to one of its own splitters or by dragging a splitter control to the desired location for the split.

   - the vertical splitter control is a short, vertical bar just to the left of the scroll bar at the bottom of a pane.
   - the horizontal splitter control is a short, horizontal bar just above the scroll bar on the right of a pane.

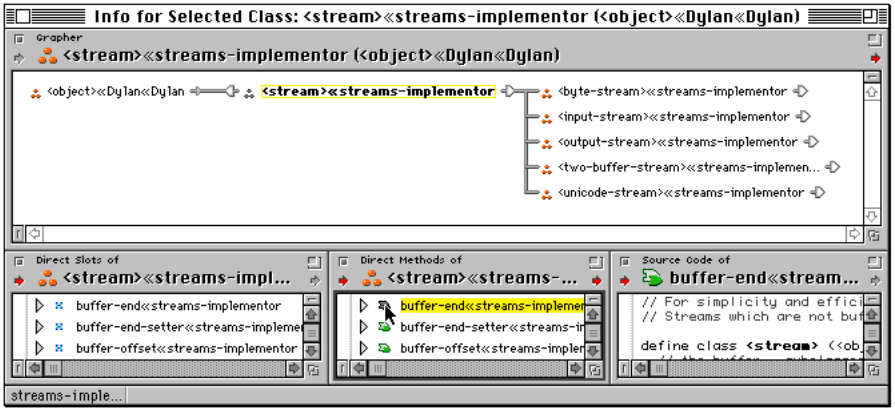
   The following figure shows the lower-right corner of the browser with the outbox of the lower-right pane being dragged onto its vertical splitter. The splitter control turns red when the outbox is on it.



   This splits the pane vertically and creates a link to the new pane. Notice the outbox is now red on the pane you split, which is half its original width, and

the inbox of the new pane to its right is also red. Nothing has been selected in the split pane, so the new pane does not display anything yet.

5. **Click buffer-end in the middle-lower pane.**
   You can now click an object in the original pane and it is displayed in the new pane. The basis for the new pane in this example is buffer-end, as shown in the following figure. The aspect Source code of is the default aspect for buffer-end, so that is the aspect displayed in the new pane.



6. **Leave this browser open as you start the next task, "Saving a browser configuration."**
   To keep a customized browser configuration for use later, you must save it before closing the browser. See the task "Saving a browser configuration" on page 61 for more information.

Saving a browser configuration

When you save a browser configuration, you give it a name, which is added to the list of browsers on the Browse menu. The new browser is also listed on the List of Browsers browser. Any changed links and aspects within the new browser are saved. Saving a browser does not save the code within it, nor any of its external links, only its internal configuration of panes.

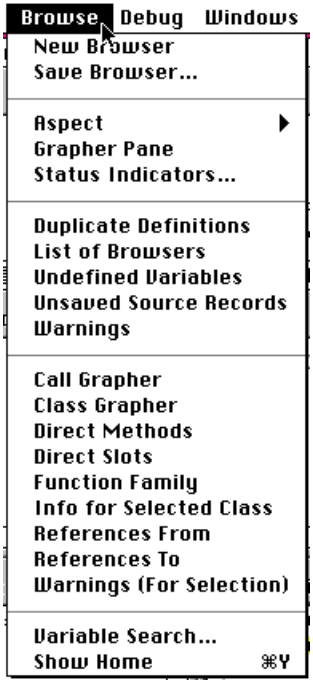
The saved browser is saved to disk as a file in the Browsers folder. If you want to share a saved browser with someone, you can send them the file. You can delete a browser from the lists of browsers available in the development

environment by moving the browser's file out of the Browsers folder in Apple Dylan. You do this by leaving Apple Dylan and dragging the file into another folder or into the Trash. The browser remains on the development environment's lists of browsers until the development environment is restarted.

1. **Select the customized browser from the previous task.**
   If you don't have the customized browser from the previous task still open, open any built-in browser from the Browse menu. You don't need to customize it to save it with another name.

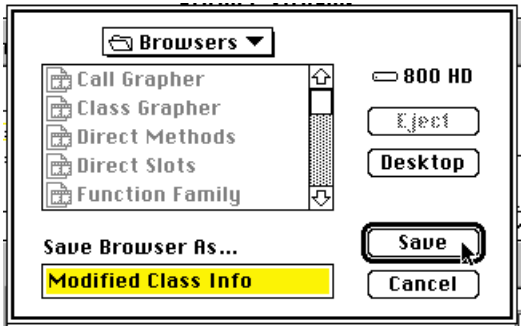
   Neither do you have to have any particular project open to accomplish this task, although in this example the Streams project is open and active. You could even create a new project by choosing New Project from the File menu. The browsers you save are available in any project you work on in Apple Dylan.

2. **Look at the Browse menu to see what browsers are currently listed.**
   The built-in browsers are in the third and fourth sections of the Browse menu. Any previously saved browsers are in the fourth section.



3. **Click in the customized browser you created in the previous task and choose the Save Browser command on the Browse menu.**
   Because the active browser is saved, be sure to make active the browser you want to save before choosing the Save Browser command. You don't have to customize a browser to save it with a new name, although that is the common reason for saving a browser.

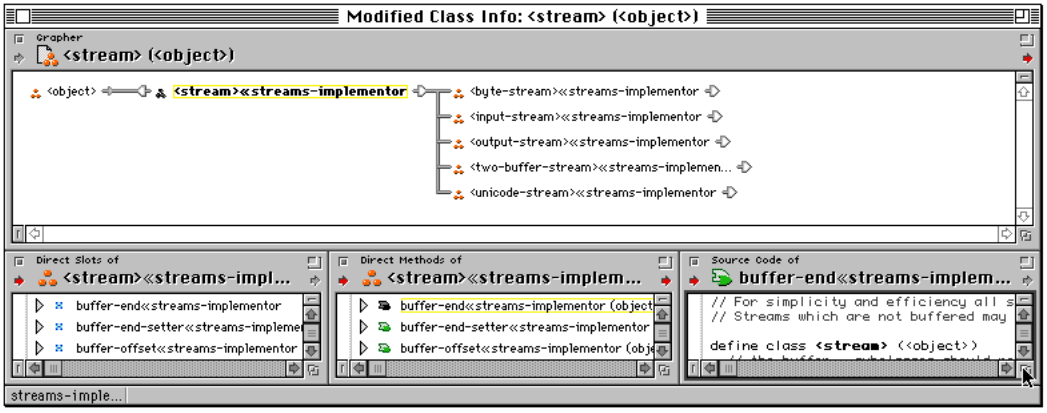
4. **Locate the Browsers folder using the dialog box, enter a name for the new browser, and click Save.**
   A browser should be saved into a file in the Browsers folder of Apple Dylan or one of its two subfolders. You can save any browser, whether an existing, new, or changed browser, and give it the name you want. If an existing browser has the same name, it is overwritten. The objects in the browser are not saved, just the configuration of panes and all links internal to the browser.



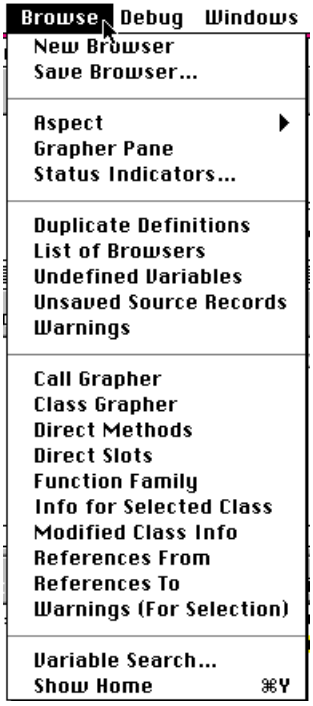
   If you save the browser to a file in the main Browsers folder, the browser appears in the fourth section of the Browse menu. This type of browser responds to a selection for its basis, such as a class. If you save the browser to a file in the subfolder _Ignore Selection Browsers, the browser appears in the third section of the Browse menu. This type of browser does not respond to a selection, but instead uses the active project as its basis. If you save the browser to a file in the subfolder _System Browsers, the new browser does not appear on the Browse menu, but only on the List of Browsers browser. You might do this if you wanted to create a new configuration for the default project browser.

   You can now copy the file, mail it to others, and handle it in many ways as any file on the Macintosh. You can open it by double-clicking it in the Finder.

The new browser is now listed on the Browse menu and probably in List of Browsers. In addition, the customized browser now shows the new name, such as Modified Class Info in this example.

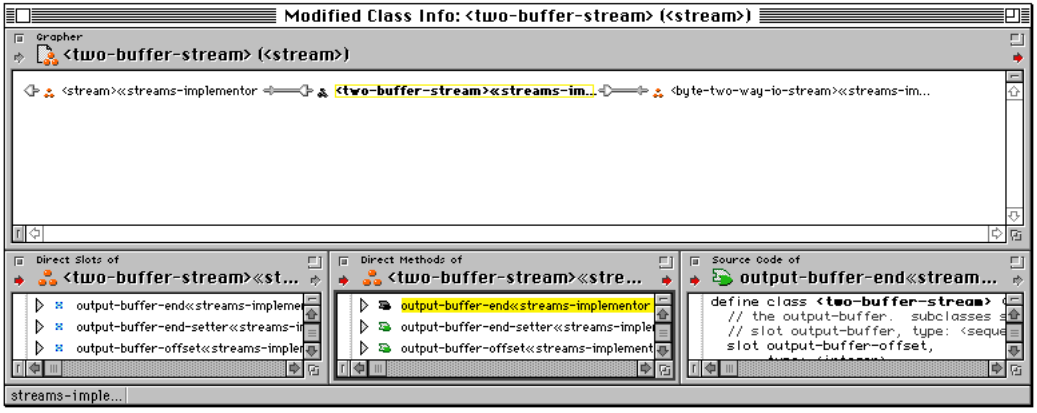


5. **Check the Browse menu to see the new browser is listed.**
   In the following figure you can see that the new browser was saved to the main Browse folder as it appears in the fourth section of the Browse menu.



6. **Close the saved browser.**

7. **Click on another class in the project browser and choose Modified Class Info from the Browse menu.**
   You see its basis is now the other class, but it has the same configuration of panes and their links.



8. In the Finder, go to the Browsers folder in Apple Dylan to see the new file.

Editing in Apple Dylan

In Apple Dylan you edit in place in the browsers. You don't have to go to a special editing window. Editing commands are available from the Edit and Text menus and work in the content area of panes. You can edit the names of objects, move them in the pane relative to one another, edit their contents, and more. Apple Dylan provides Copy Special and Insert Special commands that directly support coding in the Dylan language by creating templates for you based on existing code. Editing commands are also supported in the Listener and the New Text Window command on the File Menu.

In most cases, each source record is treated as a buffer with its own separate editor. To edit a source record, you could expand it inline using its disclosure triangle or double-click on it to open a new browser for it. You could also export the source folder it's in and edit that outside of Apple Dylan in the word processor of your choice. These files can then be imported back into Apple Dylan.

Key shortcuts provide not only common Macintosh editing commands, but also emulate Emacs-style editing commands.

Editing tools discussed in this section include:

- Copy, Cut and Paste and related Apple Dylan commands from the Edit menu
- Clear and Undo and related commands from the Edit menu
- Replace and Find and related commands from the Text menu
- Formatting commands on the Text menu
- Importing and exporting Dylan text files from the File menu
- Macintosh and Emacs-style editing key commands

In combination, these tools allow you to perform complex editing of Dylan sources while remaining within the structure of your project as displayed by the development environment.

The best way to learn editing in Apple Dylan is to do it. You can practice using the sample projects. When you are through practicing, use Revert from the File menu to bring the sample project back to normal, or simply do not save changes when you close the sample project.

▲ **WARNING**
These tools have been assembled from a variety of sources and may not behave with complete consistency. In particular, the contents of the Clipboard and the Emacs-style kill ring are not always synchronized.

Copy, Cut and Paste in Apple Dylan

The Apple Dylan development environment supports the conventional Copy, Cut and Paste commands on the keys Command-C, Command-X, and Command-V, respectively. These commands are on the Edit menu and use the Clipboard in the usual way. These commands operate on text only, not on objects.

In addition to the conventional Copy command, Copy Title Text appears when the selected object is a valid title, such as the name of a module or subproject. Copy Title Text is invoked by Command-C and also uses the Clipboard in the usual way.

Copy Special and Insert Special commands

Apple Dylan also offers the Copy Special and Insert Special commands, which are specialized for Dylan programming. These commands are on the Edit menu and support two different styles of template-based editing.

Copy Special puts a template on the Clipboard and Insert Special puts the information directly in an editor buffer without affecting the Clipboard.

These commands work only in the active project and require a valid selected object. This can be either text that names an object, such as a method name, or a source record icon. Three kinds of information are available, depending on the nature of the object selected:

- Argument List—provides a prototype argument list for the selected method
- Class Template—provides a prototype template for a subclass of the class or classes selected
- Method Template—provides a prototype template for the selected generic function or method

Copy Special appears on the Edit menu when an object is selected. The action of the command writes the chosen prototype information to the Clipboard. You can also use the key shortcut Command-J if you want just the argument list, or Command-T if you want a template. The available template changes from class to method depending on what you have selected.

Insert Special does not appear on the Edit menu, but is available by pressing the option key when you click on Copy Special. Insert Special does not use the Clipboard, but places the prototype information directly in the current editor buffer. You can also use the key shortcut Command-Option-J if you want to insert just the argument list or Command-Option-T if you want a template. The available template changes from class to method depending on what you have selected. Insert Special Argument List places the argument list just behind the insertion point, but deletes no text. Insert Special Template replaces the selected class or method with the new template.

Use these commands to get the argument list of a method you want to call, or to create new methods and classes based on existing ones. In general, the Copy Special commands will be most used, but the Insert Special commands may be helpful in some situations. Say you wanted to create a method “moo” based on the existing method “foo”. Type “foo” into the Listener and select it, then press Command-Option-T and the template for “foo” replaces the name “foo” and

you can proceed with your new method, making sure to change the name of the method in the first edit.

See the reference documentation for more information on Copy Special and Insert Special and examples of their use.

Undo and Clear commands

The Undo command is on the Edit menu. Use it to undo your last editing action. Undo is displayed as Undo Typing or Undo Format, depending on your last action. Once you have undone an action you are presented with the Redo command and the Undo More command. Undo More can undo up to the last 20 editing actions.

Undo also works with the Clear command. Clear clears, that is, deletes, whatever you have selected. If you use Clear to delete an object, folder, library, or sub-project, you cannot undo it. Undo is supported only if you have used Clear to delete text. Using Clear and Undo to delete and restore text also activates the Redo Clear, Undo Clear, and Redo commands, as appropriate in the context. Clear puts nothing on the Clipboard.

Replace and Find commands

The Replace and Find commands are located on the Text menu. Key shortcuts are provided for all these commands.

These commands work across contents of the current pane. When the target text is found, the source record containing it is expanded and the target text is selected.

Use Find/Replace (Command-F) to find and optionally replace text, using the Find dialog box. Use Find Again (Command-G) to find another instance of the target text specified in a previous Find command. Use Replace & Find (Command-J) to replace target text with replacement text and then search for the next instance of the target text.

All these actions can be performed from the Replace/Find dialog box, which also allows you to search in a single selected browser pane or throughout the entire project. The search includes any text you can display by expanding all source records. Searching the entire project is equivalent to searching the upper

left pane of the default project browser. This means that if the source code of subprojects is available, it will be searched.

Note
Most editing in Apple Dylan operates on one source record at a time, not the entire project as in this case.

In addition to the conventional Find and Replace commands, the Text menu also provides Find Selection, which searches the current pane for the selected text. Find Selection is independent of the other Find/Replace commands; it performs no replacing and does not change what Find Again searches for.

Formatting commands

In addition to the Find/Replace commands, the Text menu permits you to change the Font, Size, Style, and Color of any text to which you have write access. Changing any of these items marks the source record for recompilation, but otherwise has no semantic effect. There are no established guidelines on text markup, but it is common to use **bold** for the name of the object being defined. Use text markup to highlight whatever distinctions are convenient for you and your project, such as assigning colors to individual code, or particular type sizes or styles to different functions. Autostyling is not available.

Text markup from Apple Dylan is exported and imported in .dylan text files. Text markup imported from other editors is ignored.

Importing and exporting Dylan text files

Apple Dylan supports the exporting and importing of Dylan text files. Dylan text files have the suffix ".dylan" and are a text version of a source folder (not a source record). Dylan text files can be mailed or edited outside of Apple Dylan.

Dylan text files can be created in a number of ways:

- Select the source folder to export and use the Export command from the File menu
- Use the New Text Window command from the File menu
- Use most text editors or word processors outside Apple Dylan and save the file with the .dylan suffix. Import the file into Apple Dylan.

The Export command performs only limited code checking. Export breaks a single source record with more than one top-level form or comment into an individual source record for each top-level form or comment when exported.

You can bring Dylan text files into the Apple Dylan development environment using the Import command.

You can export a project with partial source records, but you will not be able to reimport it. If any exported code within a source folder has the incorrect number of begins or ends, or is any other way seriously malformed, the source folder cannot be imported.

Importing or exporting has no effect on objects marked for exclusion or inclusion.

Macintosh and Emacs-style editing commands

Apple Dylan supports editing using both standard Macintosh key commands and Emacs-style key commands. In most cases you can intermix these commands, keeping in mind that the contents of Clipboard and the Emacs-style kill ring may be inconsistent when using both.

Macintosh-style key commands

The Macintosh-style scrolling commands (Page Up, Page Down, etc.) do not move the insertion point. Once you get where you're going, you must click to move the insertion point. The Emacs-style cursor-movement commands move the insertion point.

The following table summarizes the Macintosh-style editing commands. In all cases, a source record is a buffer and vice versa.

cursor motion within a buffer	forward	back
character	←	→
word	Option →	Option ←
down & up line	↓	↑
end & beginning of line	Command →	Command ←

cursor motion within a buffer	**forward**	**back**
down & up page	Command ↓	Command ↑
last & first line of source record	Command-Option↓	Command-Option↑
end & beginning of source record	Command-Option→	Command-Option←
motion across buffers	**forward**	**back**
next & previous open definition, move line by line through contiguous expanded records	Option ↓	Option ↑
expand & enter source record below & above.	Command ↓	Command ↑
expand & enter source record below & above while closing current.	Command-Option↓	Command-Option↑
buffer -> object		
select enclosing object	Control ↑	
select enclosing object & collapse	Control-Option↑	
motion object -> buffer,		
expand and edit selected source record	Command ↓	
motion at object level	**forward**	**back**
select source record	↓	↑
select last or first source record	Option ↓	Option ↑
expand & collapse source record	Command ←	Command →
page forward or back	page down	page up
bottom or top	end	home
select 1st at new level (expanding if needed)	Control ↓	Control ↑
end or Beginning of buffer, if already there, expand next	Command ↓	Command ↑
select 1st at new level and expand or collapse	Control-Option↓	Control-Option↑
beginning or end of buffer, if already there, expand next and collapse previous	Command-Option↓	Command-Option↑

Emacs-style key commands

The Emacs-style support is provided by a partial implementation of the Fred editor (Fred Resembles Emacs Deliberately), which originated in Macintosh

Common Lisp. Fred is included for the convenience of those who already know Emacs.

Assigning meta to the option key, which you can do using the Preferences command, may conflict with some of the Macintosh key commands using the option key. In such cases, use Control-Q followed by the option key to insert a literal option key.

Meta-X commands are not implemented. The meta key must be pressed and released for each use, not held down like the Control or Command keys. Where the meta key is used in combination with other modifying keys, it works best to press the meta key and release it before pressing the rest of the key combination.

The Fred kill ring contents may sometimes overwrite the Clipboard. Clipboard contents are not available from the Fred kill ring.

In addition to the usual use of the mouse to select text by dragging across it, two clicks selects a word, three clicks selects a line, and four clicks (tricky) selects the entire buffer, that is, the entire source record.

You may discover other Emacs-style commands in Fred, but their employment is not supported as they may produce ambiguous or erroneous results.

The table summarizes Emacs-style editing commands. This table is not intended as documentation of these commands, but as a checklist of those Fred commands that should work in Apple Dylan.

The meta key is assigned to escape by default. You can choose to use the option key as the meta key through the Editing Category on the Preferences dialog. This setting of the meta key conflicts with using the option key for Macintosh-style editing commands. In this case use Control-Q to quote the option key as itself.

Emacs-style Keys	
cursor motion	
• Control-B, ←	Move the insertion point back one character.
• Control-F, →	Move the insertion point forward one character.
• Meta-B, Meta←	Move the insertion point back one word.
• Meta-F, Meta →	Move the insertion point forward one word.
• Control-A	Move the insertion point to the beginning of the line.
• Control-E	Move the insertion point to the end of the line.
• Control-P, ↑	Move the insertion point up one line.
• Control-N, ↓	Move the insertion point down one line.
• Meta-V	Scroll upward.
• Control-V	Scroll downward.
• Meta-<	Move the insertion point to the first line of the source record.
• Meta->	Moves the insertion point to the last line of the source record.
• Meta-M	Moves the insertion point to the first non-white-space character in its current line.
Selection	
• Shift ←	Move insertion point one character to the left and select.
•Shift →	Move insertion point one character to the right and select.
• Meta-Shift ←	Move insertion point one word to the left and select.
• Meta-Shift →	Move insertion point one word to the right and select.
•Control-Shift-A	Move insertion point to beginning of line and select.
•Control-Shift-E	Move insertion point to end of line and select.
•Control-X H	Select entire buffer and move to beginning.
•Shift ↑	Select to same point on previous line and move insertion point.
•Control-Shift-P	

	Emacs-style Keys
•Shift ↓	Select to same point on next line and move insertion point.
•Control-Shift-N	
•Control-Shift Space	Set mark.
	Insertion
•Control-O	Insert new line but don't move insertion point.
•Control-Meta-O	Split line and indent.
•Control-Y	Yank current kill ring. Replace selected text if any.
•Meta-Y	Rotating yank.
•Control-Q	Insert next keystroke quoted—use if Meta key assigned to Option and you need Option.
•Meta "	Insert pair of double quotes around insertion point.
•Meta-U	Make rest of word uppercase and move insertion point to end.
•Meta-L	Make rest of word lowercase and move insertion point to end.
•Meta-C	Capitalize first letter of rest of current word or selection and move insertion point.
•Control-T	Transpose two characters surrounding insertion point.
•Meta-T	Transpose two words surrounding insertion point.
•Command-click	Replace the selected text with the Dylan expression you command-click on. Especially useful for replacing one name with another, less useful for other Dylan expressions; Dylan syntax is not well understood by the Fred editor.
	Deletion
•Delete	Delete character to left of insertion point.
•Meta-delete	Delete word to left of insertion point.
•Control-D	Delete character to right of insertion point.
•Meta-D	Delete word to right of insertion point.
•Control-K	Delete remainder of line.

	Emacs-style Keys
•Control-W	Delete current selection and add to kill ring.
•Meta-W	Copy current selection and add to kill ring.
•Control-X •Control-Space	Delete all spaces and tabs from insertion point to next character.
•Meta-Space	Replace all spaces and tabs surrounding insertion point with a single space.
•Meta-\	Delete all whitespace characters to left and right of insertion point.
	Undo
•Control-_	Undo previous command.
•Control-Meta-_	Display Undo history.
	Numeric arguments
•Control-U	Repeat next keystroke 4 times.
•Control-*n*. Meta-*n*	Repeat next command *n* times.
	Incremental search
•Control-S	Initiate forward incremental search.
•Control-R	Initiate reverse incremental search.
•Delete	Delete last character typed from search string.
•Control-G	Clear search string.

Editing code

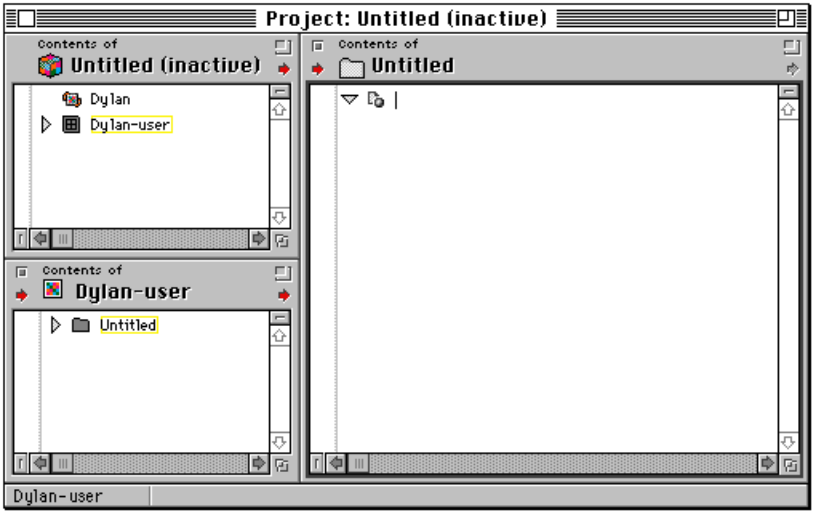
You can edit code and rearrange objects in a browser. You can drag objects between projects to copy them or copy code from individual source records between browsers or projects. You can also copy code to and from the Listener or a text file.

Several sample projects have been included with Apple Dylan. You can use code from them by copying it into your project. In addition you can use code from the framework, whose source code has been included in Apple Dylan.

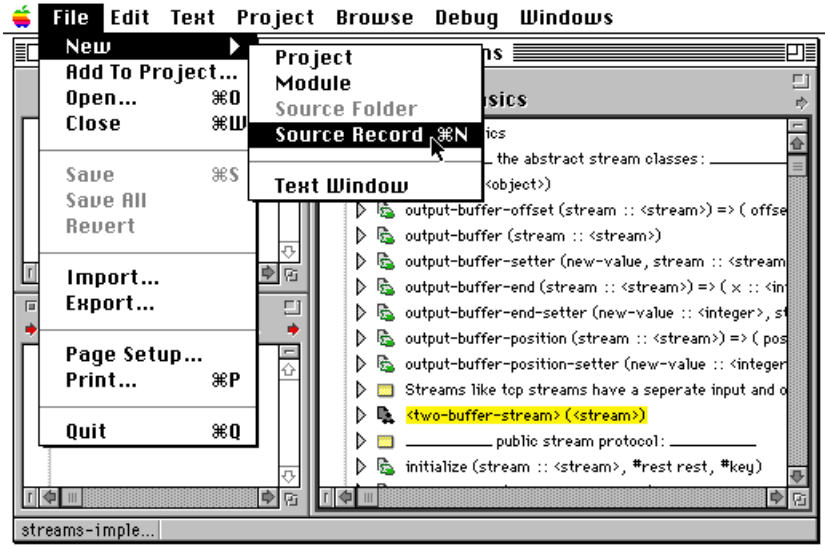
You can edit code in any project, it does not have to be the active project. However, you cannot compile code in anything but the active project.

1. **Open the sample project Streams, if it's not already open, and make it the active project.**
   The sample projects can be found in the Sample Code folder or by using the alias in it.

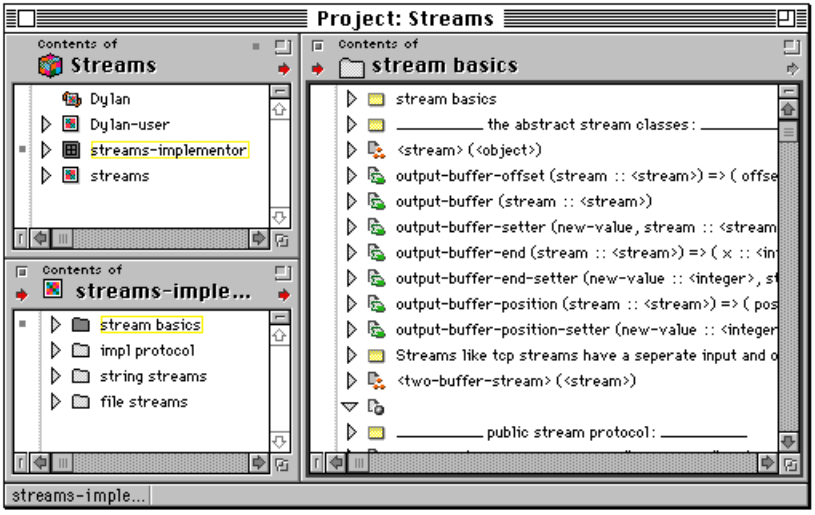
2. **Create a new project using the New Project command on the File menu.**
   The following figure shows the default new project browser. Notice that the Dylan-user module is automatically selected, an untitled source folder automatically created within it and an empty source record within that. You could type or copy code into the source record, but for now click in the Streams project so you can work in it first.



3. **Select a location for a new source record in the Streams project and choose New Source Record from the File menu.**
   Select an object in the pane you want the new object in. This is how you add modules, source folders, and source records to an existing project. In the following example the class <two-buffer-stream> has been selected in the Streams project browser and New Source Record chosen from the File menu.



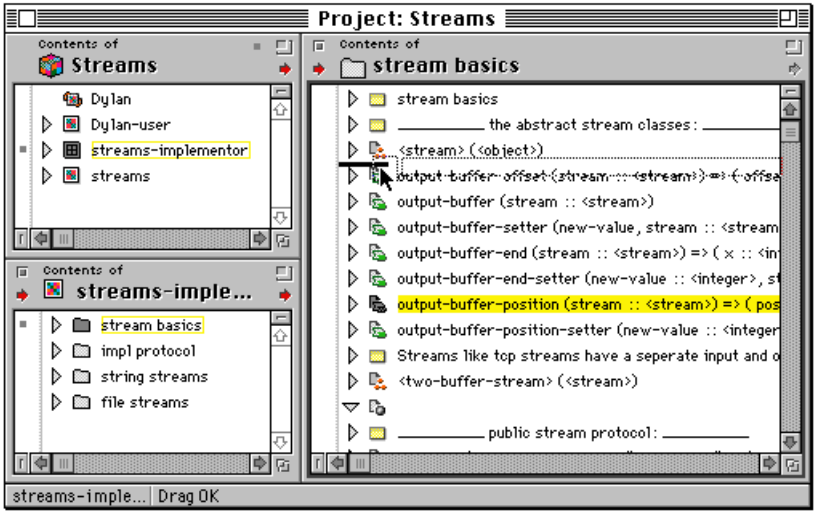
The new source record appears after <two-buffer-stream, as shown in the following figure.



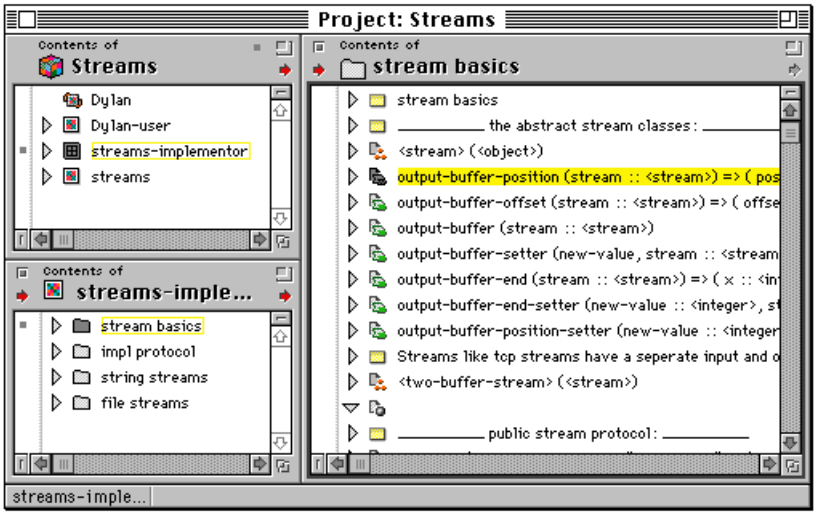
You could type or copy code into the new source record. To copy code, simply highlight the code you want to copy, whether it's in another source record, even in another project, or in the Listener and then use the Copy command.

4. **Select the icon of the source record output-buffer-position and drag it above output-buffer-offset.**
   This is how you can rearrange objects in Apple Dylan. You can also drag objects between panes, between browsers, and between projects. Dragging between projects copies the object, dragging within a project moves it.

   Select the module streams-implementor and its source folder stream basics, if the source record output-buffer-position is not visible. In the following figure, notice the heavy black marker that indicates where the dragged source record will be inserted when the mouse is released.



When you release the mouse, the source record appears at the position of the heavy black line. In the following figure the drag has been completed and the source record output-buffer-position is above output-buffer-offset.

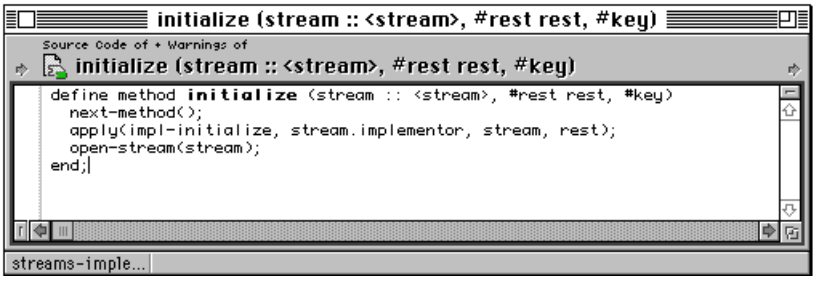


5. **Expand the source record <two-buffer-stream> so you can see its source code.**
   You can edit its source code inline using the Edit and Text menu commands.

6. **Select text within the expanded source record to delete and choose Clear from the Edit menu.**
   The text is deleted with the Clear command.

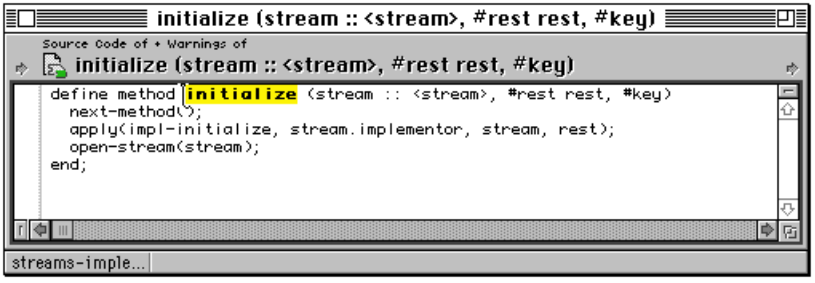
7. **Choose Undo Clear from the Edit menu.**
   To reinstate the text, use Undo Clear. Notice that the exact wording of the Clear and Undo Clear commands change appropriately to match the type of action you can perform. You can undo up to 20 editing commands.

8. **Double-click on the source record initialize.**
   Although you could edit an expanded source record inline in the project browser, you will probably want to edit the code in a separate browser. You can edit this code using the Edit and Text menu commands. The following figure shows the new browser that opens with initialize as its basis.

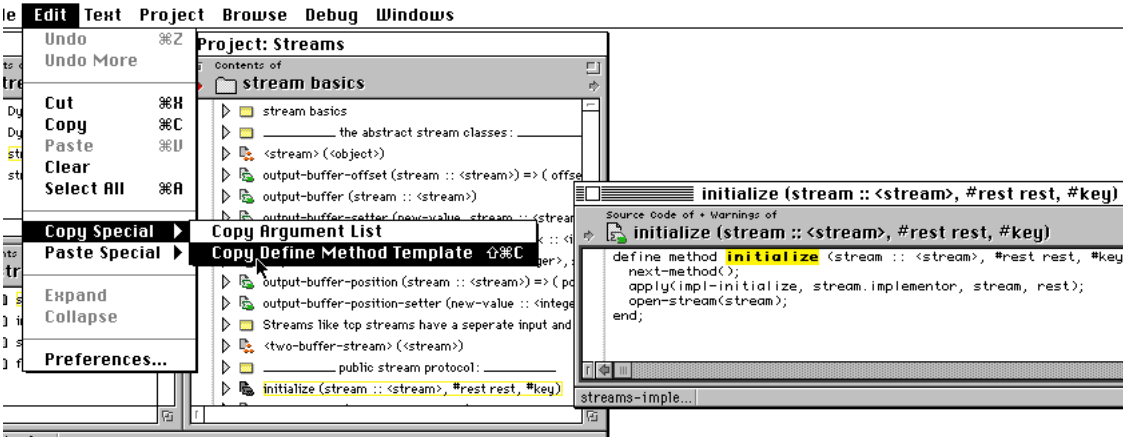


9. **Select the name of the initialize method in the new browser to use it as a template for a new method.**
   The Copy Special and Insert Special commands use the language databases within Apple Dylan to retrieve code templates.

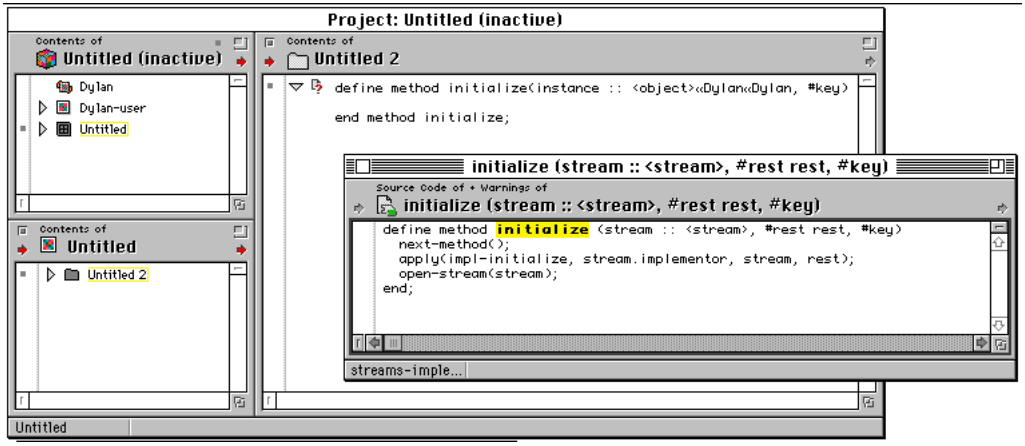
   In the following example, initialize has been selected and will be used as the basis of the template.



10. **Choose the Copy Special command Method Template from the Edit menu.**
    The Method Template (Copy Special) command copies the template to the Clipboard. The following figure shows initialize selected and Method Template being chosen from the Edit menu.



11. **Go to the source record where you want to use the template, click where you want the code placed, and choose the Paste command.**
In this example, go to the newly created source record in the new project and click in the new source record. In the following figure the code template for initialize has been pasted into the new source record in the new project. You can see both the new project browser with the new code in its right-hand pane and the separate browser in front of it with initialize still selected in it. Click in the new source record to edit the template and create your own version of initialize by specializing at least one of the parameters.



The Insert Special commands work in a similar way to the Copy Special commands, but they do not use the Clipboard.

12. **Choose Close from the Project menu to close the Streams project.** You can also use the close box on the project browser to close the project. Don't save the changes to the Streams project. Close your new project, saving it if you wish.

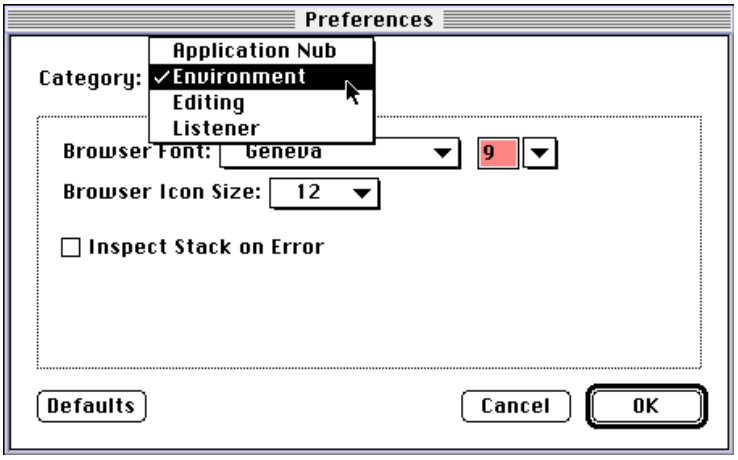
    There is no need to have any projects open for the next section because you set the preferences for the development environment as a whole, not for individual projects.

Customizing the development environment

You can customize your projects in several ways, such as through the subprojects you choose to incorporate. In the development environment, you can also customize how you view and interact with all your projects. You can choose various settings through the selections you make in the Preferences sheets. You don't need to have any projects open to complete the tasks in this section as the preferences you set apply to all projects.

Several categories of preferences can be set, including the font size of the environment itself, text editing conventions within each pane, interaction between the compiler and your code, and interaction between the runtime while you are debugging your code. You make these choices using the Preferences command on the Edit menu.

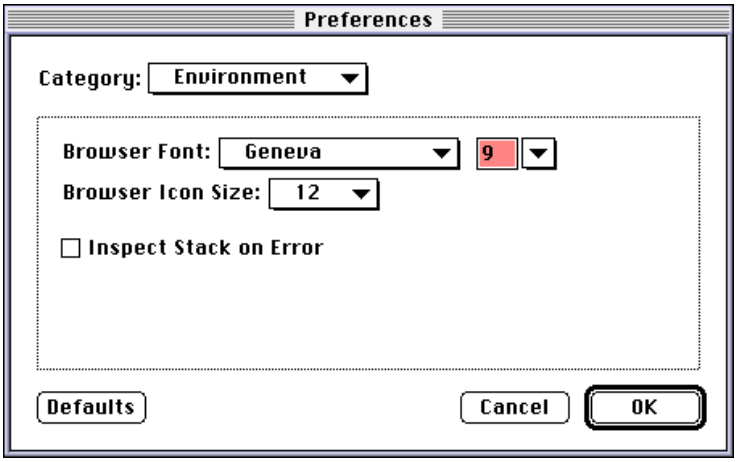
The Preferences command presents four dialog boxes, or sheets, for customizing the interface of the Application Nub, Environment, Editing, and the Listener. Choosing the Default button resets the defaults for all fields on all four sheets.



Setting development environment preferences

You can customize your development environment so that you can work with every project in the same way. By setting fields in the Environment sheet, you can choose the font for browsers, the font size and icon size for browsers, whether to automatically open the Stack window if you get an error, whether you automatically launch the application nub whenever the active project is opened, and whether you want to automatically update the active project whenever the application nub is launched.

1. **Choose the Preferences command from the Edit menu.**
   The Environment sheet opens. The following figure shows the default values for the Environment sheet. Choosing the Default button resets the defaults for all fields on all three sheets.



2. **Choose the type of font you want used for display in the browsers.**
3. **Choose the size for icons.**
   The icons in the development environment are clearer if you choose an icon size of 16 in the Browser Icon Size field.
4. **Click Inspect Stack on Error if you want to open the Stack window on any error that occurs while compiling your code.**
   See the chapter "Using Apple Dylan" on page 95 for more information on the implications of this choice.
5. **Click Launch Application Nub when Active Project is Opened if you want to launch the application nub whenever you open a project.**
   See the chapter "Using Apple Dylan" on page 95 for more information on the implications of this choice.
6. **Click Update when Application Nub is Launched if you want to update the active project whenever you launch the application nub.**
   See the chapter "Using Apple Dylan" on page 95 for more information on the implications of this choice.

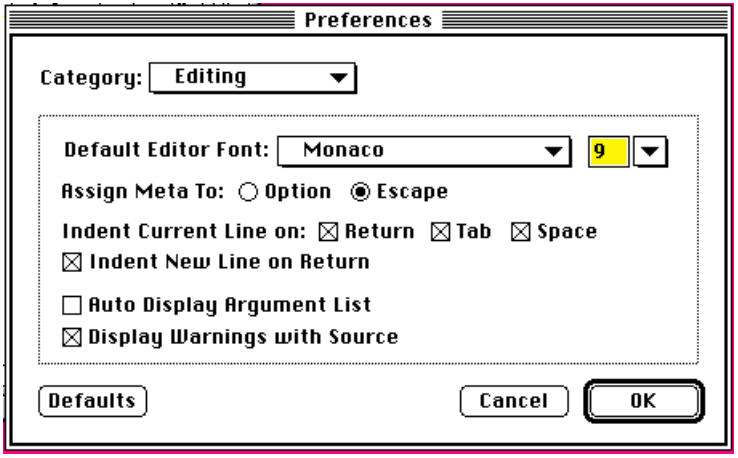
Setting editing defaults

You can customize your development environment so that your editing environment is the same whenever you edit code. By setting fields in the

Editing sheet, you can choose the font to use when editing code, the meta key you want to use, if any, how to automatically indent code, and whether you want to display warnings with the source code.

The following figure shows the default values for the Editing sheet. Choosing the Default button resets the defaults for all fields on all three sheets.

1. **Choose the Editing sheet from the Category popup.**
   The following default sheet appears.



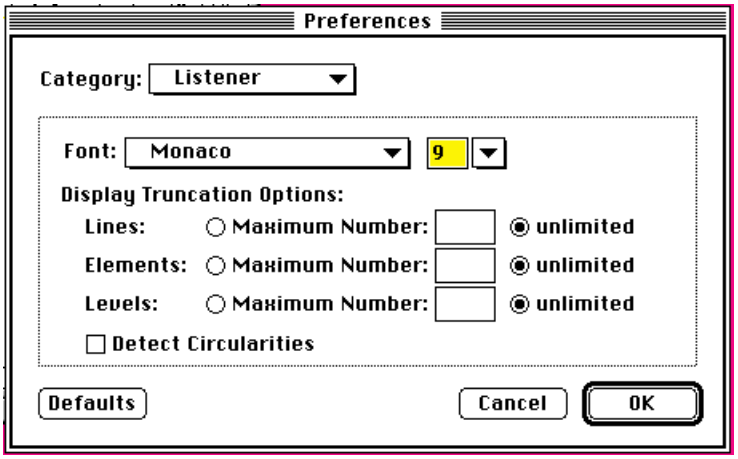
2. **Choose the font family and size you want to use while editing code.**
   Changes you make in these fields apply only to edits you make after changing this.
3. **Change the meta key for editing with the Emacs-style key commands, if you want.**
4. **Click in the boxes for the type of indentation you want applied to your code.**
5. **Click Auto Display Argument List if you want to automatically display the argument list.**
6. **Click Display Warnings with Source if you want to automatically display any warnings within source records.**

Setting Listener interaction defaults

You can customize the way the Apple Dylan Listener acts. By setting fields in the Listener sheet, you can choose the font for the Listener, what truncation of lines, elements or levels you want, and whether you want to detect circularities.

The following figure shows the default settings for the Listener sheet. Choosing the Default button resets the defaults for all fields on all three sheets.

1. **Choose the Listener sheet from the Category popup.**
   The following default sheet appears.



2. **Choose the font family and size you want to use while editing code in the Listener.**
3. **Choose a maximum number of lines you want printed in the Listener, if you want to limit them.**
4. **Choose a maximum number of elements of a list you want printed in the Listener, if you want to limit them.**
5. **Choose a maximum number of break levels you want displayed in the Listener, if you want to limit them.**
6. **Click Detect Circularities if you want to stop printing after the first time around a code circularity.**

Using icons in Apple Dylan

Every object depicted in the Apple Dylan development environment has an icon associated with it. Icons are also used to convey other kinds of information in Apple Dylan.

For each object, there can be three kinds of icons:

- First, there is the basic icon of a definition entity, in this case the definition entity of a class that is the compiled representation of a class:

- Next there is the icon of a source record for a class, the basic icon superimposed on a standard document icon. This is the most commonly encountered form:

- Finally, there is the icon of an excluded source record, the basic icon superimposed on a standard comment icon:

- The ball icon indicates a generic object:

- Thus, the following is a generic source record:

- A **?**, signifying a problem of some sort with the object: this may be superimposed on other icons.

You control the display size of icons through the Environment sheet of the Preferences command. See “Setting development environment preferences” on page 84.

- default size, 12 points
  :

- 16 points

- 32 points

Following is a summary of the most commonly encountered Apple Dylan icons, displayed in their most commonly encountered forms:

- Project

- Subproject, the same as project

- Project File, document with .π suffix, seen in Finder

- Generic source record

- Application seen in Finder. The Application Nub and Apple Dylan have this icon.

- Comment

- Variable source record

- Constant source record

- Macro source record

- Database source file

- Note icon, usually no match. Usually accompanied by an explanation of what has not been matched, such as “Inapplicable aspect”.

- Unrecoverable Error

- Saved Browser

- Top level form

- Warning, recovery may be possible.

- Source folder

- Source module

- Class source record

- Library file, .dl suffix.

- Bare method source record

- Generic function method source record

- Generic Function source record

- Source database in Finder

- Compiler results database in Finder

- Resource file

Close any open projects when you are done with this chapter. Don't save the changes to the sample projects.

CHAPTER 2

Using Apple Dylan

The Apple Dylan development environment is powerful and easy to use. You can create applications quickly because the environment is organized around objects, just as the Dylan language is. You will see that the various types of code elements appear in the environment with different icons so you can tell them apart. These objects can be manipulated by dragging and dropping, just as you would suppose.

Apple Dylan User Model

The Apple Dylan development environment is built around a number of assumptions, procedures, and expectations about how programming is done in Apple Dylan. These ideas and requirements are known collectively as the Apple Dylan user model. The user model is a description of the process you use to create applications and libraries in Apple Dylan, not a description of the internal implementation of Apple Dylan.

This section introduces the user model by means of a walkthrough of the programming process, starting with organizing and writing source code and continuing all the way through to building a stand-alone application or library. Detailed descriptions of how to complete various tasks follow this overview.

The Project

The project is the central concept in Apple Dylan. A project is a set of documents that encompass all the elements of your programming effort:

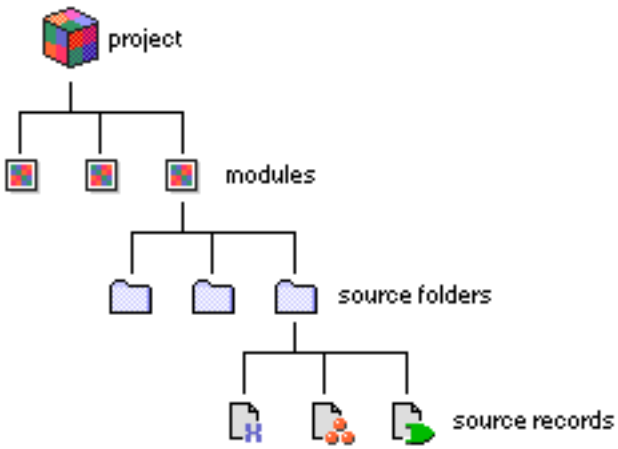
- libraries
- subprojects

- modules (sometimes called source modules)
- source code
- resource files
- text files associated with the project

The Apple Dylan development environment allows you to examine and change all of these program elements.

The container hierarchy in Apple Dylan is as follows, top-down:

- Projects contain modules
- modules contain source folders
- source folders contain source records
- source records hold source code.



In essence, a project is made from source records, individual definitions represented as individual objects that can be directly manipulated by the development environment or edited as text. Most of the other objects in Apple Dylan are means of organizing the source records. For instance, a module is the place where all your definitions are kept, usually within source folders. The source folders are simply a convenience for grouping and ordering the source records.

You can add libraries to a project, just as you can add any project to another. When you add them to a project, the added libraries and projects are represented as subprojects in it. These subprojects, along with any resource files you add, appear in the project at the same level in the hierarchy as modules.

A library is a namespace for module names. They allow you to use other people's libraries without causing name collisions between modules. Libraries are created with the "define library" statement. A module is a namespace for variables, so you can similarly avoid name collisions between variables when using other people's libraries. Modules are created with the "define module" statement.

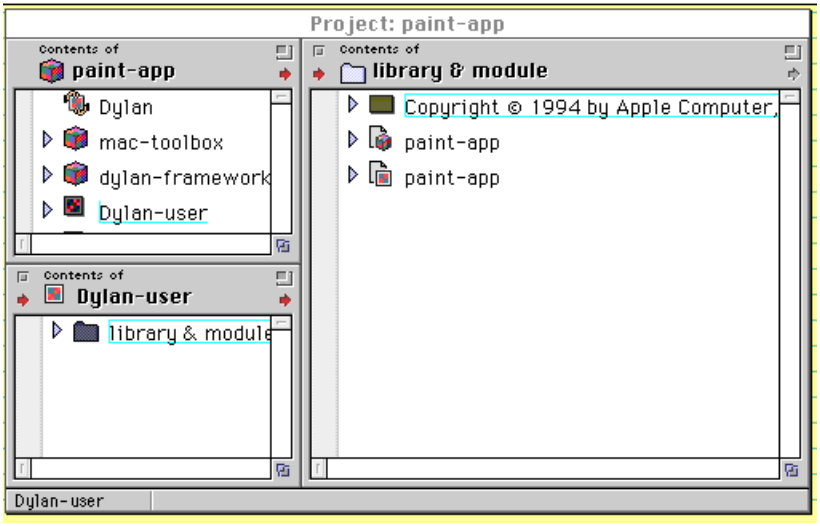
The end result of your work on a project is either an application or a library. An application runs when you double-click it. A library is a building block used by an application, allowing for sharing of reusable code.

A number of sample projects are included in the Apple Dylan distribution. These sample projects are used for all examples in this document as well as being available for independent learning.

What goes into a project?

When you start Apple Dylan, the first thing you see is the Listener. You can't do much with the Listener without an active project open, so open a sample project, such as "paint-app", using the Open command. Open the file in the Apple Dylan folder with the path Sample Code: More Samples: paint-app: paint-app.π. (The π is the suffix that identifies the document as a Dylan project.)

You should see the default project browser open. The Dylan-user module is selected in the root pane in this example:



The root pane of the project browser (upper-lefthand pane) displays the major elements of the project. Each project contains one or more subprojects, which contain the libraries that the project's library needs. Every library needs the "Dylan" library, one language library, one or more modules contained by the library (each library contains an implicitly defined "Dylan-user" module), and any text or resource files, if needed.

Here is what you see in the root pane of a typical project's project browser (you might have to scroll or zoom the pane to see all its contents):

- Dylan library—a subproject that contains the Dylan language itself. This subproject looks different from other subprojects because it has no source code. All projects include this library.
- mac-toolbox—a subproject that contains a set of import statements in Dylan that give access to the (non-Apple Dylan) routines in the Macintosh Toolbox.
- dylan-framework—a subproject that contains an object-oriented class library that implements (in Apple Dylan) a common set of features found in Macintosh applications.
- Dylan-user—a module defined for each project. The module is intended primarily for setting up other modules and libraries, but is useful in small or experimental projects as a single module.

- paint-app—an additional module containing the code specific to this application. There can be any number of additional modules, or none at all.
- paint-app.rsrc—a resource file used by the application. This file is optional. You may have several resource files and they need not be kept in the same folder as the project.
- For some projects, you will also see other subprojects listed in the root pane. A subproject is a project or library that has been included in another project. Any project whose project type is "library" (see the Set Project Type command) can be included in another project.
- For some projects, you may also see text files. It may be convenient to store these files—perhaps a README, notes or lists—with the rest of the project, but the development environment does little or nothing with these files.

Good practice requires that every project have a library definition. The library definition makes modules available for use. This is done with a `define-library` statement in the Dylan-user module.

That a module or library/subproject name is displayed in a browser means that it is included in the compiler results database, but it does not mean that it is available to the application. Only clauses in a project's define library statement makes modules from other libraries accessible to the project's modules. The `use` clause must be typed in. It does not appear automatically.

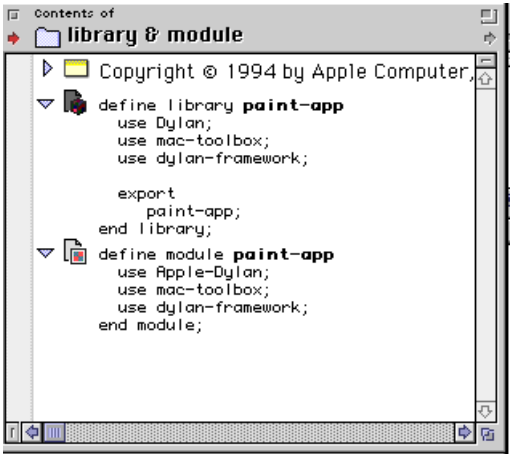
Each variable within a module is visible to all the code contained in that module, but variables must be explicitly exported (and then imported) to be visible to code in other modules.

When you make a new module using New Module from the File menu, you are informing the development environment of the module. When you add a `use` or `export` statement to your source code, you are informing the compiler of the module. You must do both because the integration of environment and language are not seamless in this release. This is one of the seams.

There must be a single library definition including a `use` statement for each library and an explicit statement exporting the module that contains the startup function for an application.

There must also be one or more module definitions, showing which modules are accessible from that module and thus allowing access to all exported variables.

To see these definitions, move to the righthand pane of the project browser and expand the two paint-app objects by clicking on their disclosure triangles:



It is here that the scope of the project is defined, that is, which libraries and modules form part of the project.

Note
In this example, the module definition includes a `use` statement for Apple-Dylan. If you were writing portable code for use in non-Apple environments, the `use` statement would name Dylan, not Apple-Dylan so the compiler would catch any use of Apple extensions.

Note
The Dylan-user module is in each project, but you must define for each project a unique version of the Dylan-user module. It is possible to write a program without defining a library or any other modules by simply using Dylan-user for everything. With the exception of the tiniest experiments, this practice is not recommended as it circumvents the design of the language and the development environment. When you create a new project, you should create a new module or modules to hold all your code.

What's *really* in a project?

Use the Finder to examine the directory for the paint-app sample project. You'll find the following on your hard disk:

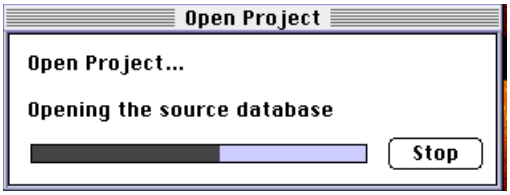
- paint-app.π—the project file, which actually consists of pointers into other files. This is the file to open when you want to use a project in the development environment.
- _Source Database—all source records for the project, in database rather than text form.
- paint-app.rsrc—an optional resource file. Use ResEdit or another resource editor to change its contents. There may be no resource file, or many; they need not be in the project folder.
- paint-app.dl—the project library file.
- _Compiler Results Database—all compiled objects for the project.
- _Library Model—equivalent to the compiler results database, but for libraries. Created by Create Library.
- paint-app (the executable file)—the standalone application itself, which can be double-clicked in the Finder to run paint-app.

Under normal circumstances you never do anything directly with any of these documents. The main reason to mention them is to point out that you have the option, in the case of a severe need for disk space, of deleting _Compiler Results Database and _Library Model. These documents are created when you compile your project and thus can be replaced. The document _Source Database cannot be replaced.

Note that these two files, _Compiler Results Database and _Library Model, grow along with your project but do not shrink along with it. As you repeatedly compile, old compiler results are not eliminated, but simply cut off from access. You may find it valuable from time to time to issue the command Compact Project to perform a garbage collection on these files.

What happens when you open a project?

You open a project using the Open command on the File menu. When you do, you see an Open Project dialog box like this, giving you status as the project is being opened:



Here is what happens:

- All files in the project are locked against other access and opened.
- A project browser is created.
- The development environment checks for an active project and if there is none, makes this the active project. Otherwise, this project is opened but is not the active project.
- The source database index is loaded.
- The compiler results database is loaded (as the dialog box mentions loading of caches and definitions).
- If you have selected "Launch Application Nub when Active Project is Opened" on the Preferences Environment sheet, the Application Nub is launched.
- If you have selected "Update when Application Nub is Launched" on the Preferences Environment sheet, the Update Project command is issued.

The active project

Only the active project can be browsed completely. Only one project, the root project, can be active at one time, although all subprojects contained within that project are also active. No other project, open or closed, is active. The first project you open is the active project, but you can change active projects with the Activate Project command from the Project menu. If you close an active project, the project opened next becomes the active project.

Other open projects are inactive. You can browse the text portions of these projects, but you cannot browse any relationships that depend on compiler results. The source records of an inactive project can be edited and saved. You can also cut and paste between active and inactive projects. Changed text in an inactive project is marked as uncompiled when it is changed, but you cannot compile it until you make it the active project.

Only the active project and its subprojects can be fully browsed. You could think of the development environment as an interface to the compiler results database. In other words, all information displayed by a browser depicts what has actually been compiled. For instance, the call graphers and class graphers depict calling or class structures that have been successfully compiled, not what has been incorrectly written in the code and won't compile. The same goes for all cross-references, debug information, or other relationships in the project. Keep in mind, you are working directly on your application or library, not a representation of it.

Because each source record is a definition, individual definitions can be edited, and compiled without disturbing the rest of the program. Thus, in compilation, only modified definitions are compiled. This incremental compilation means that you can interactively change the active project with a very short compile time.

Note
The changes you make in the active project are not reflected in the compiler results database until you compile them. Therefore, many browsers do not reflect the changes until you compile.

▲ **WARNING**
Since the compiler does not lock files, you are not prevented from editing while compiling. However, you absolutely should not do this. You should also not browse while compiling.

Targeting 68K and PowerPC Platforms

You can target your final application or library to run on the Macintosh 68K or PowerPC platforms. You set an individual target architecture while you are developing a project. To do this, you make your project active and choose the preferred platform using the Target Architecture command. Then, when you build your final application or library, you can choose which architecture you want it to run on. If you want a single application to run on both architectures, you would choose to create a **fat application**. A fat application should run equally well on both platforms as it contains native code for both. You can also

choose to create a **skinny application**. A skinny application is one that runs native on the 68K architecture but is emulated on the PowerPC.

Although you can switch back and forth between the two targets for any active project as often as you wish, you must issue an Update Project command every time you do so, which can slow development time. Therefore, if you want to develop for both platforms, you can decrease the number of times you have to switch between the two platforms by getting your project fairly solid for one before compiling it for the other. The common methodology is to target one of the platforms, develop your application or library through the debugging stage, then choose the other platform and recompile. You don't have to recompile your project with the other architecture targeted. You can instead create a fat application and test that.

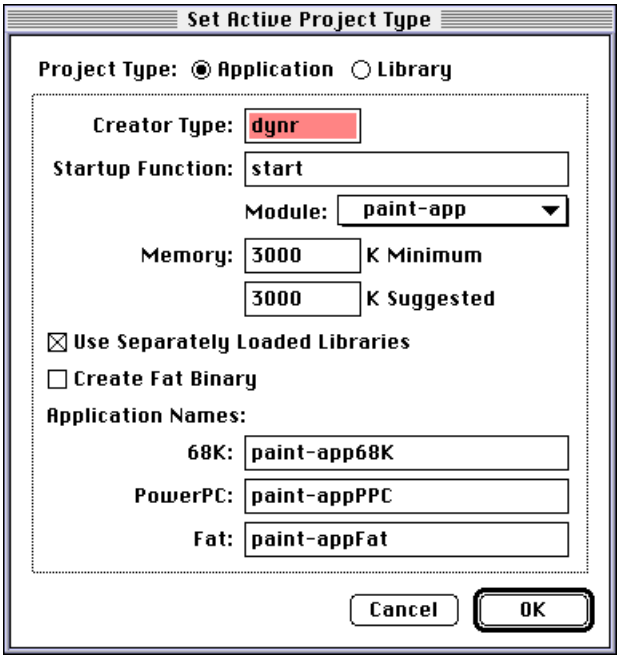
The reason you must use the Update Project command every time you switch between platforms is that there is a separate compiler results database file and library model file for both platforms. These files are updated only for the current target; the other set of files are left as they were when you switched away from them. Therefore, an update is needed to refresh the contents of these files when you switch back to them.

Application or library?

Every project must have certain characteristics established, depending on whether it is an application or a library. This is done through the Set Project Type command from the Project menu. You can also use this command to examine the characteristics of any project.

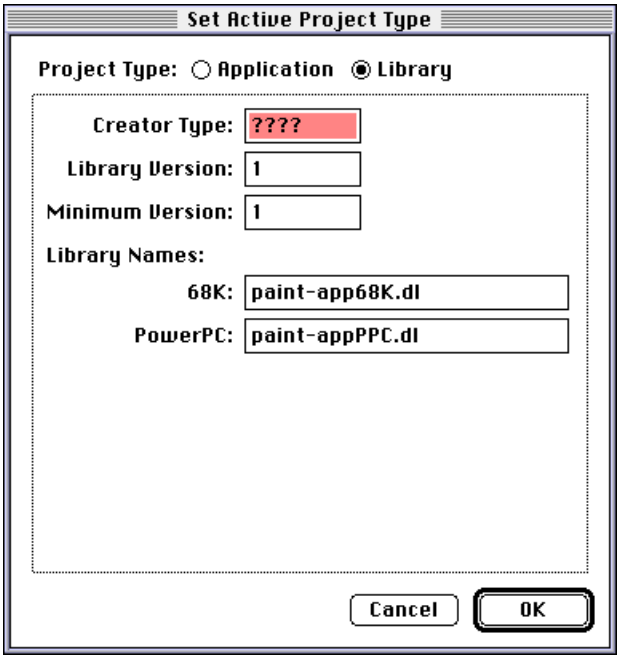
Internally, applications and libraries are identical, with the single exception that an application includes a startup function. This means that an application can be started and achieve some end result. A library does nothing by itself and must be used as a building block of an application

For the sample project paint-app, you see the following dialog box when you choose Set Project Type from the Project menu:



This information is needed by the development environment to manipulate, load and run the application. You can see the paint-app project is an application. The startup function (similar to Main in a C program) is identified by name and location (i.e., which module it's in). The file creator type is given, as are the suggested and minimum memory needs of the application. The box "Use separately loaded libraries" is checked.

If you check "Library" in the "Project Type" field, the dialog box changes, reflecting the differences between applications and libraries:



There is no startup function in a library, nor does a library have memory requirements. Also, libraries do not vary according to whether they are separately loaded or not. However, libraries do require versioning.

Library version numbers

Set Project Type allows you to specify a version and a minimum compatible version for each library.

Setting the version of a library controls which version is recorded in the compiler results database. When a library file is created from a project, the library versions of all of its subprojects are recorded in the library file header. Then at runtime, when the library is loaded, it will only accept those versions of the sublibraries that are compatible with the version of the sublibrary used at library creation time.

The minimum compatible version is the lowest version that a library is backward compatible with. The Min Version field is checked during runtime library search.

For example, suppose library A uses library B, and at the time the library file for A is built, library B was at version 17. Later, at runtime, when library A is being loaded, it will look for library B, version 17.

- If it finds an instance of library B that has version 17, that version is used.
- If it finds an instance of library B that has version 16, that version is not used.
- If it finds an instance of library B that has version 19, and the minimum version is 17 or less, that version is used, because even though that library has a later version, it claims to be backward compatible with version 17.
- If it finds an instance of library B that has version 19 and a minimum version of 18 or more, that version is not used, because that library is not compatible with version 17.
- If it finds more than one instance of library B which is compatible with version 17, it picks one at random.

If library B version 17 had a minimum version of 16 at the time Library A was built, it might seem that version 16 of Library B could be used, but it doesn't work that way. If library B has version 17 and minimum version 16, that means only that version 17 is backward compatible with version 16. Thus, version 17 could be used where version 16 was expected. However, it doesn't mean that version 16 is compatible with version 17. Library A could be using some new features of library B version 17 that weren't in B version 16. Thus, version 16 cannot be used for libraries built on version 17, even if the minimum version is 16 or less.

Note
In other words, there is no way to specify that a library is backward compatible.

The Application Nub

The Application Nub is a minimal application that communicates with the development environment. A project under development can be loaded into the Application Nub and run under the control of the development environment.

The Application Nub also permits you to debug a standalone application (one that has already been built and runs outside of Apple Dylan).

Your application cannot execute unless the development environment is tethered to either a standalone application or the Application Nub. The development environment can communicate either with a project loaded into the Application Nub or with any standalone application that has been built using Apple Dylan, providing you have the compiler results database for the standalone application. If so, you can run the application from outside of Apple Dylan, then return to Apple Dylan and download code to the application, insert breakpoints, insert `print` statements, use inspector windows, and test the project or portions of it using the Apple Dylan Listener. You have full access to such a standalone application, just as if it were still a project being run from within Apple Dylan.

In either case, tethered means tethered to the runtime, which is the actual running state of your project or application, and connected to the Apple Dylan Listener, which informs you of all values returned and permits direct interaction with the running application.

Tethering the development environment to the Application Nub or a standalone application adds no overhead. All debugging information is included in the compiler results database and the source database, not in the standalone application or the Application Nub. All projects and shipped applications are fully debuggable, simply by tethering them to the development environment.

Keep in mind that any development activity that is supported for a project loaded into the Application Nub while it is tethered to the development environment is also supported for any Apple Dylan application for which you have the compiler results database. That means that an important part of preparing an application for distribution is saving a copy of the compiler results database for the standalone applications you build.

Anything compiled while tethered to the development environment is automatically downloaded to the runtime. The compiler results database is always synchronized with the runtime while tethered.

Two closely related commands support tethering to the development environment:

- Launch Application Nub from the Project menu launches the Application Nub, loads your project into it, and tethers it to the development environment.
- Tether to Application assumes that a standalone application is already running and tethers the development environment to it.

Neither Launch Application Nub nor Tether to Application downloads any code to the Application Nub. Compiling code with a compilation command, such as Compile Selection or Update Project, does that.

Launch Application Nub has these effects:

- Starts the Application Nub running.
- Establishes communication between the development environment and the Application Nub.
- Initializes the runtime.
- Loads the Dylan library.

Note
Launch Application Nub does not run the project!

Tether to Application has only a single effect:

- Establishes communication between the development environment and a running standalone application or the Application Nub.

Note
The standalone application must already be running when you issue this command! You also must have the compiler results database for the standalone application.

You have the option on the Environment page of the Preferences dialog box from the Edit menu, to automatically launch the Application Nub when you make a project active. You also have the option of performing an Update Project whenever a project is launched. These options are off by default and independent of each other.

You are not limited to using the Application Nub supplied with Apple Dylan. You may also add to it to create a custom Application Nub of your own. This enables you to include object libraries from other languages or other resources from outside the domain of the Dylan language, such as Creole files.

Keeping your project synchronized

There are three sets of information for your project that must be kept in synchronization through your own actions. They are:

- source database—source records in text form as presented by the development environment, which can be changed by means of editing a source record and saving it.
- compiler results database—all code that has been compiled and related debugging information.
- runtime—all code that has been downloaded to the Application Nub.

Ordinarily, Apple Dylan allows you to switch smoothly from the runtime to the databases. A typical scenario would be to observe a problem in the running application that is attached to the Application Nub, use a browser to find the problem, and then use the editor to change the source and correct the problem. If you follow this activity with an Update Project command, your databases will be synchronized.

However, there are many opportunities for your information to get out of synchronization. Moving or deleting Apple Dylan files while not in Apple Dylan can have bad consequences, as does renaming any folders. The development environment will not be able to find the object and report it as missing. Missing files need to be found and identified when the development environment is running or they will just be ignored during compilations. When they have been identified, the project must be recompiled and saved. While the development environment is not running you can move the entire Apple Dylan folder or a project's entire folder and it will be OK.

Sometimes you may face an even more confusing situation, such as when you have coded an object and compiled it, then deleted the source. Then, when you use a browser, you might find the object is still there. This is because the browser is working off the compiler results database, not the database of source records. The running application attached to the Application Nub can show yet a third behavior because you may have downloaded a compiled definition that is no longer active. Further, if there are warnings or if there has been a downloading error, your code may have been compiled without being downloaded. While the Update Project command usually corrects desynchronization, it is up to you to keep in mind that you are working across three databases and that they do not necessarily reflect the same objects.

Note
Keep in mind that compiler results are saved to the compiler results database each time you compile anything, but the source database only changes when a source record or the active project is explicitly saved with the Save or Save All command, respectively.

Orphan definitions in the runtime

One form of desynchronization may be particularly confusing, objects that exist in the runtime, but have no apparent source. These are called orphan definitions and come from three causes:

- the source of the definition has been deleted or renamed
- the object was defined in the Listener and compiled, downloaded and executed from there
- the definition is in a library, and is consequently downloaded with the library when you launch the Application Nub.

In all cases, the result is a definition entity in the runtime that has no corresponding source record. The best means of eliminating these objects is to untether from the development environment, retether, and then issue an Update Project command.

Note
You can also eliminate an orphan definition by using an Inspector to inspect the function and using the Inspector's remove command. However, this can lead to the reverse situation, also confusing, where you have an object defined in the source and compiler results databases that is not present in the runtime.

Status indicators and synchronization

The best information on whether your databases are synchronized comes from the status indicators. At the left edge of each pane you can see a vertical gray stripe, sometimes with tiny squares in it. This is the status indicator bar. You can choose which kinds of status the bar shows for each pane, or whether it shows anything at all.

Choose which status indicators you want active for each pane by selecting the pane and then choosing Status Indicators from the Browse menu or by double-clicking on the status indicator bar. The Status Indicators dialog box opens. The more status indicators you choose from it, the wider the status indicator bar in the pane. The status indicators you can choose are:

- Unsaved—blue square: the object or its contents have changed since the last time it was saved or it has never been saved. You clear the indicator by saving.
- Uncompiled—green square: the object or its contents include source code that has changed since the last compilation or has never been compiled. You clear the indicator by compiling.
- Warnings—orange square: when the object or its contents were compiled, the compilation resulted in warnings. You clear the indicator by fixing the problem and recompiling, if it's a compilation warning.
- Read Only—red square: the object or its contents cannot be changed. You cannot turn the read-only attribute off or on from within the development environment; it is set by other applications.
- Other—dark gray square: the composite category of unchosen indicators. If Other is the only status indicator chosen for the pane, then any object that is unsaved, uncompiled, has warnings, or is read-only will be flagged with the Other indicator. If Other is chosen along with one or more other status indicators, then Other stands for the unchosen indicators. If all other indicators are chosen, Other is not listed on the Status Indicators dialog box.

The status indicators appear in the order named, with Unsaved farthest to the left and Other farthest to the right. The status indicators are color coded as noted. If you click on a status indicator in a pane, the Prompt Area at the bottom of the browser identifies it. If the indicator is Other, the Prompt area shows the current definition of Other.

You can also see the squares representing status indicators in the header of a pane, if you choose. These show the status of the basis of the pane.

Restoring synchronization

The status indicators show what is out of synchronization. It is easy to bring your project back into synchronization, but there is no single command that saves your changed sources, compiles everything that needs compiling and

downloads everything to the runtime. Saving sources is always a separate activity.

Save sources with Save or Save All from the File menu. Save is available only when you have explicitly changed the source in a selected object. Save All is always available. Revert from the File menu returns you to the sources as they existed when you last saved the project.

Update Project compiles all uncompiled or changed sources. If the project is loaded into the Application Nub while it is tethered to the development environment, the newly compiled sources are downloaded to the running application.

The Run command combines several functions:

- tethers to the development environment, if needed
- performs an update, compiling all sources marked as uncompiled and downloading them to the Application Nub, if needed
- runs the project's startup function.

The actions performed by Run may vary according to the state of your project and can't necessarily be determined in advance.

The Recompile command recompiles all sources, whether marked as uncompiled or not.

Apple Dylan Listener

The Apple Dylan Listener is a window into a running application. Type a function call in the Listener and its results are printed in the Listener. This is the fundamental Listener interaction. The Listener is not an interpreter or a command-line window. In the Listener you are tethered to the runtime, not the development environment. You are compiling code for the actual application, downloading to it, and inspecting the actual results.

The Listener is similar to the MPW shell except:

- The Listener executes Apple Dylan code, not shell scripts.
- The Listener distinguishes what you have typed by displaying it in bold.
- The Listener is sequential and historical in that it supports interaction primarily at the last prompt position and always prints results at the last position, while the MPW shell allows interaction anywhere in the shell and prints at that spot.

While browsers allow you to look at the compiler results database and the source database, the Listener allows you to interact directly with the runtime. Inspector windows allow you to look at objects in the runtime. See the section "Inspector windows" on page 116 for more information.

The Listener title bar includes "(Unconnected)" when the development environment is not tethered to the Application Nub. Once you have launched the Application Nub (or tethered to a running standalone application), the Apple Dylan Listener is connected to the Application Nub and is, therefore, available. The Listener allows you to interact with the application as it runs. In fact, you can run an application by executing its startup function in the Listener.

Note
If the Listener prints no returned values, or prints only warnings for syntax errors, you are not tethered to the runtime.

By having the Listener open as you program, its feedback from listening to your actions helps you monitor your progress. You can enter any Dylan expressions into the Listener for immediate execution. For example, you can call functions in the Listener (including your application's startup function), and you can define functions, variables, and classes in the Listener.

The Listener is useful for quickly testing expressions without creating containers to hold them. In addition, the Listener provides access to parts of your application that you cannot access any other way, such as data stored in a database or off a hash table. The Listener is inherently temporary, similar to a scratch pad. If you wish to make permanent changes in your project, you should do so in a browser, saving, compiling, and downloading the changes from there.

The form typed into the Listener and then compiled, results in zero or more values when executed. These values are printed in the Listener, one per line. For example, executing `values ()` results in no values, so none are printed. Executing `1 + 2` results in `3`, which is printed on the next line. Executing `values ("first" "second")` results in two values that are printed on the next two lines.

All values returned or any other results are printed to the Listener window for any code executed by the application or anywhere in the development environment, even in the browsers. The results of macro expansion are also printed in the Listener.

You should launch the Application Nub before compiling code from the Listener. You can do this by using the Launch Application Nub command on the Project menu. Although you can compile without being tethered to the runtime, all the compiler does in that case is check the code's syntax.

Code typed into the Listener is compiled and downloaded to the application under development. The objects created appear in browsers and behave in most ways as if they were a permanent part of the project, but they are not. When you compile and download to the Application Nub from the Listener, you are creating a definition entity for the object, but you are not creating an associated source record. Although every object has one or more associated definition entities, definition entities created in this way are temporary, lasting only as long as you remain tethered to the runtime. They disappear when you untether. The Listener listens, but does not remember. The Listener is not a browser. When you code in the Listener, you must use the editing tools to move useful source text into a source record in a browser to make it permanent. See the section "Orphan definitions in the runtime" on page 111 for more information.

▲ **WARNING**
Nothing created in the Listener is saved. You should not create methods and classes in the Listener even though it is possible to do so. Use the browsers instead; they are designed for that purpose.

The Listener is a place to test and compile code without permanently modifying the project. It is useful for ephemeral code testing, for running of individual functions to see what they do, and for inspecting and changing the state of the runtime.

Before you type an expression into the Listener, choose the module the expression is in from the popup list in the lower-left corner of the Listener window.

▲ **WARNING**
Failing to choose the proper module in the Listener is perhaps the single most common and confusing mistake Apple Dylan users make. Remember, each module is a namespace and all code is executed in the context of a module. If you are not in the proper module, you are not in the proper namespace and the Listener will literally not know what your are talking about.

You can execute code from the Listener while you are in a break loop, allowing you to investigate various results. When in a break loop, the Listener's prompt changes to the number of the break level on the stack, starting with the first break loop as number one.

When the Application Nub is not executing code, it is in the Listener loop, waiting for code to be sent from the development environment. This can be from compiling something in the development environment by means of one of the compilation commands or by means of typing code into the Listener. The effect is the same; the code is executed and the result values are sent back to the Listener. A break loop is the same as the Listener loop, except that there is code waiting to be executed that was stopped by an error or Break and is waiting to be aborted, inspected or resumed. The Listener loop can be thought of as break level 0 or the null break level. For more information on break loops, see the section "Debugging a project" on page 146.

The Listener supports three forms of interacting with your project or application.

- If you observe behavior in your application that you wish to change, you can enter a Break command to suspend the application while you make your changes. All state is preserved, including the stack.
- If your application returns an error, it will be suspended and a backtrace supplied that allows you to determine the exact source of the error.
- If you press command-option-. (command-option-period), the application is suspended exactly as it is at that moment, without losing any state or stack information. You are still in the middle of the application and can see exactly what it is doing.

Inspector windows

While browsers allow you to look at the source database and the compiler results database, and the Listener allows you to interact with the runtime, inspector windows allow you to look directly at objects in the runtime. To use inspector windows, you must be tethered to the application and the application must be suspended.

The inspector windows offer your only opportunity to look directly at objects in the runtime. In fact, objects only exist in the runtime, and furthermore many objects in the runtime are created by the running application.

Several commands that open inspector windows are included in Apple Dylan:

- The most common use of inspector windows is through the Listener, where you can compile, download and execute code, look at the results as printed in the Listener, and then use the Inspect Listener Result command to determine exactly what happened.
- After an error, the stack is presented in the form of a backtrace. Use the Inspect Stack command to inspect the state of the program at the point of the error.
- You can list all the modules in the Application Nub and select one from the list to open an inspector window on it using the Inspect Module Variables command.
- You can also look at your memory allocations and contents using the Inspect Heaps command.
- Finally, you can select any object and inspect it using the Inspect Selection command.

You can use inspector windows on any object in the runtime. To do so, you must know where the object is located, such as a list or vector or other data structure.

Note
You can open another inspector window from any inspector window by double-clicking on an object in it. This can lead to having numerous inspector windows open at once. If you hold down the Option key while closing one inspector window, all of them close.

Bailing out of Apple Dylan

Sometimes you might have an error or other problem that locks up Apple Dylan to the point where you cannot proceed. You might need to quit the application or the development environment, but you cannot. In that case, you need a way to forcefully exit, or bail out.

The common method of stopping an application is by making it the frontmost and typing command-option-period. If this stops the application, you can often continue to issue other commands, such as Quit Application to officially quit the application and the Application Nub, or the Quit Application Nub command, as needed.

Sometimes you need to forcefully stop the Application Nub itself. If you have tried the commands Quit Application and Quit Application Nub, yet the Application Nub's status indicates it is still running, you might need to run the Quit Application Nub program from the Finder. You can do this by leaving the development environment and launching the executable file Quit Application Nub, which is in the Application Nub folder. The Quit Application Nub executable's purpose is to seek out and destroy any Application Nubs that are running, but it doesn't affect Apple Dylan.

When you cannot quit Apple Dylan itself, using command-period or command-option-period should stop Apple Dylan so you can use the Quit command. Usually, using command-shift-escape also lets you exit safely. However, sometimes this might hang your machine. In case these don't work, it might be useful to create an AppleScript program that tells Apple Dylan to quit. The script is:

```
tell application "Apple Dylan" to quit
```

Building standalone applications

When you have completed coding and testing your project, you can turn it into a standalone application or library. This is the final stage of the Apple Dylan user model. Only applications created in Apple Dylan can be maintained, revised, or supported using Apple Dylan. The standalone application you create is identical to the project loaded into the Application Nub, except that it is no longer connected to the development environment. If you need to change the standalone application in any way, however, you can tether to it, return to the use of all the facilities of the development environment, and then recreate it as a new application. To do this, you must have retained the project's compiler results database file. See "What's really in a project?" on page 101 for more information.

You create an application using the Create Application command. This command first untethers the Application Nub—prompting for confirmation—and then takes the information in the compiler results database (and associated libraries) and packages your project as an application that is then written out to a specified location on the disk. At that point, the project is no longer in the Apple Dylan environment. The Application Nub has been included in the application.

You create a library using the Create Library command. This command first untethers the Application Nub—prompting for confirmation—and then takes the information in the compiler results database (and associated libraries) and packages your project as a library that is then written out to a specified location on the disk.

This completes our walkthrough of the Apple Dylan user model. The rest of this chapter describes some detailed tasks and examples using the development environment.

Starting a project

When you want to write your own application or library, you create a new project using the New Project command. The new project will consist of numerous other objects, most of which you will create. Your project can also contain subprojects, which are other projects or libraries you include in your project.

Projects consist of modules which contain source folders. The source folders contain source records. A source record contains the source code, each source record being one method, class, variable, constant, macro, generic function, comment, or top level form. For more information about creating a user interface for your application, see the book *Creating a User Interface in Apple Dylan*.

You can create and open as many projects as you want, but only the active project can be compiled or browsed. You can designate any open project as the active project using the Activate Project command. Also, closing the active project with other projects open makes the next frontmost of the open projects the active project. It also closes all the active project's subprojects, if they aren't also subprojects in another project that's still open.

You also use the New Project command to create a new subproject. If the project you want to use as a subproject already exists, you simply add it to your project with the Add to Project command on the File menu. When the project it is included in is not open, the subproject is simply a project again, so it is best to think of a subproject as a type of project.

Creating a new project

You create a new project using the New Project command on the File menu. When you create a new project, you must decide how you want its code organized in its modules, source folders, and source records. You must decide if it is going to be an application or a library, if you want to develop your project with the runtime on a separate machine, and what Macintosh architecture you want your final standalone application to run on. You must also decide what C libraries you want to include in it and which subprojects or libraries you want to add to it. Also consider if you want to use it as a subproject in another project.

Each new project automatically has certain objects in it. The Dylan subproject contains everything defined in the Dylan language and the Apple Dylan extensions. The Dylan-user module is created for each project, but its contents must be defined by you. The Dylan-user module will contain the module definition and library definition used to structure the rest of your project. You must write your own module definition and library definition in this module. These two language definitions are described in *Programming in Apple Dylan*, *Apple Dylan Extensions and Framework Reference*, and the *Dylan Reference Manual*, which are shipped with Apple Dylan.

In the development environment you can create your modules using the New Module command from the File menu. You can have as many modules as you want, although one module is sufficient in a project. Modules hold source folders. New modules are placed at the end of the list of modules. You can reorder the modules by dragging them. Changing the module order changes the load order. In the Dylan language, forward declarations do not create load order restrictions. However, load order dependencies, such as calculations that are performed in one module based on a variable previously defined in another module, must be accounted for in the module order you establish. For more information on load order, see the *Dylan Reference Manual*.

Source folders are for organizing your code. Source folders hold the individual source records. You can have as many source folders as you want and most projects have several.

A source record is an individual method, class, variable, constant, macro, generic function, comment, and top level form. You can have as many source records as you want.

When you create modules, source folders, and source records, each will automatically get its own type of icon. A generic icon is assigned to each object as you are creating it. When you have completed enough of the object for the development environment to determine what it is, the appropriate icon for it appears automatically.

In addition to these objects that are in every project, you can add several others, if you want, using the Add to Project command on the File menu. The mac-toolbox subproject is a library containing the Macintosh toolbox calls that you can add to your project. It is up to you to include any or all of the individual Macintosh toolbox calls you want to use in your project.

The dylan-framework subproject is a library containing the Apple Dylan application framework. You must add the framework to your project if you want to use it. Since the source code for the framework is included in its subproject, you could modify it if you want. However, this is not a common practice, nor is it recommended.

Another type of file you can add to your project is a resource file. You must create whatever resource files you need outside of Apple Dylan using your favorite resource file editor. The resource file you add to your project will have the suffix ".rsrc" and be in the root pane of your project.

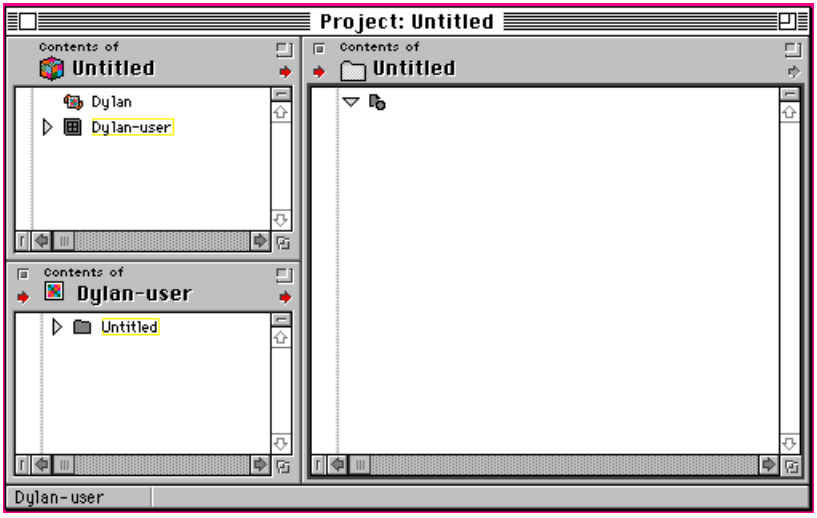
If you want to investigate Macintosh toolbox calls, you select a call and choose the Look Up in Online Reference command. The Macintosh Programmer's Toolbox Assistant is launched, if you have purchased it and it is present on your system. The reference entry for the call is displayed. You can also display reference entries for the Dylan language and the Apple Dylan extensions to it, including the framework, by selecting the function and choosing the Look Up in Online Reference command.

1. **Choose New Project from the File menu.**
   When you enter a name for your new project, you see a new project open with a three-paned, project browser. The name of the project appears in the browser's title bar. Untitled is the default name until you save it with a name.

   You do not have to close any open projects when you create a new project. However, if there are other projects open already, be sure you set the active project to the project you want to compile. Any open, inactive projects can be edited, but not compiled or fully browsed. Use the Activate Project command on the Project menu to make this the active project.

   In the root (upper-left) pane are the two default objects, Dylan and Dylan-user. The Dylan-user module contains an empty source folder and

source record. In the following example, the Dylan-user module has been selected in the root pane so you can see its default contents.



Name the new project using the Save All command on the File menu.

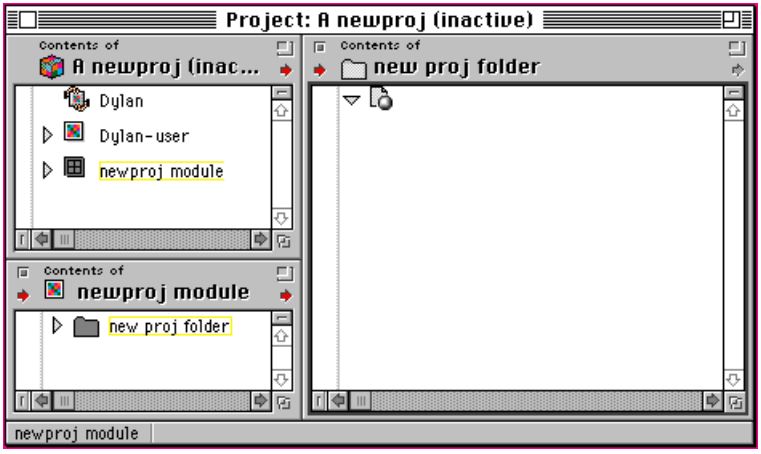
2. **Select the Dylan-user module in the root pane and choose New Module from the File menu.**
   Name the module by typing its name into the dialog box. The new module appears in the root pane below Dylan-user because you selected Dylan-user. You can create as many modules as you want, but often a single module is sufficient.

3. **Select the new module and then choose New Source Folder from the File menu.**
   An empty source folder appears, for which you can type in a name. You can create as many source folders as you want.

   The new source folder has an empty source record in it. When it is newly created, you can type the name of the new source record next to its icon at the prompt. When you enter code, the name of the source record is derived from the text of the code. If you want to rename the source record later, you must expand it using its disclosure triangle and edit the code.

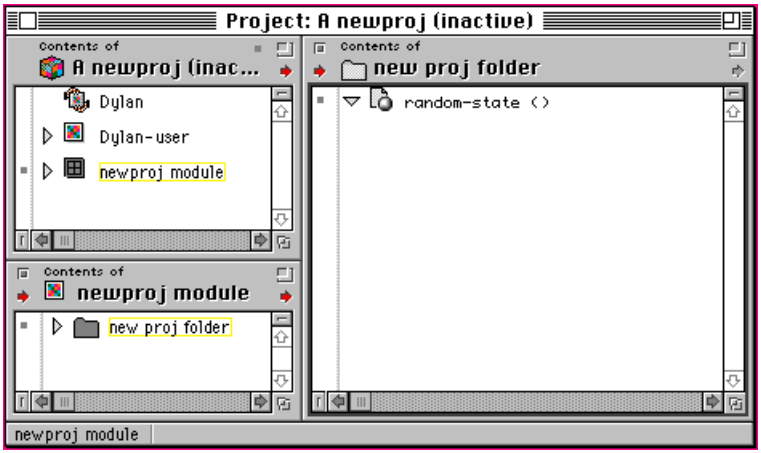
4. **Select the new source folder and then choose New Source Record from the File menu.**
   The new source record appears at the end of any existing source records unless you select a source record prior to creating a new one, then it appears after the selected source record. You can create as many source records as you want.

   The following figure shows a new source record that has been created in the "new proj" source folder of the "newproj" module. The new source record is still unnamed in this example. Because no code has been entered yet, the new source record is represented by the generic icon.

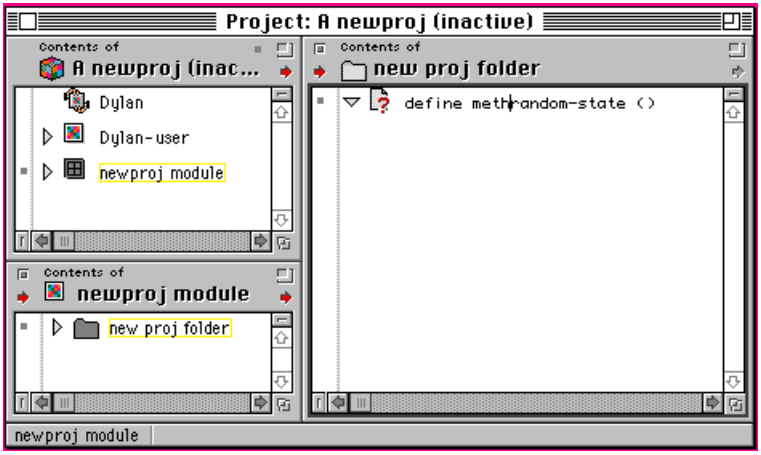


5. **Type or copy source code into a new source record.**
   There are several types of source records, including methods, generic functions, comments, and constants. Each has a different icon, which appears when enough code has been entered for the development environment to decide what it is. You edit the name of a source record, which is based upon the text of its code, by expanding it with its disclosure triangle and editing the code.

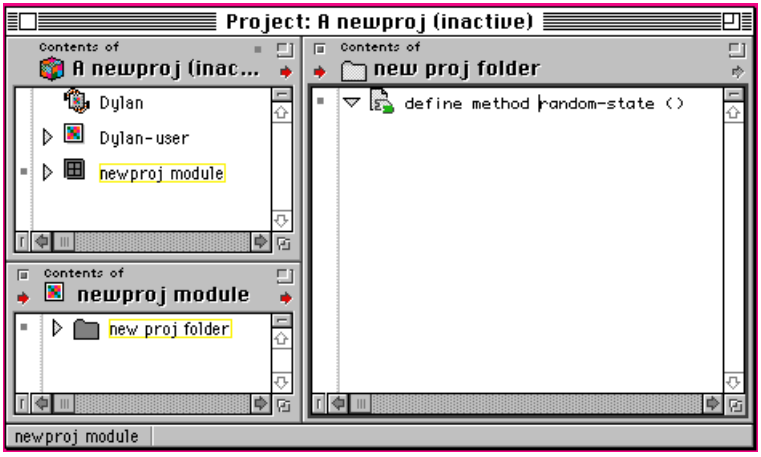
   The following figure shows a new source record named random-state. It still has the generic icon because only its name has been entered. Notice that the source record random-state has been expanded to show that no source code has been written.



The following figure shows the source record random-state with its code partially entered. It has the icon that represents a source record of an unknown type because not enough code has been entered for the development environment to decide what it is.

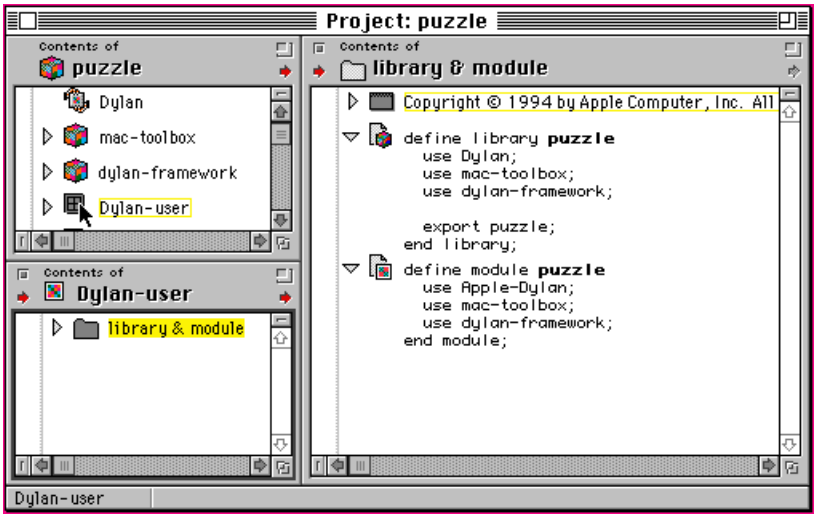


The following figure shows the source record random-state with its method icon. Notice that the entire contents of the method are not needed for the development environment to recognize it as a method.



6. **Click Dylan-user in the root pane.**
   It is empty because you must define its contents. Usually you use one source record to define the library for your project, and another to define your module. Check the *Programming in Apple Dylan*, *Apple Dylan Extensions and Framework Reference*, and the *Dylan Reference Manual*, along with the sample projects, for specifics on how to write this code.

   The following figure shows the definitions in the Dylan-user module for the sample project puzzle.



7. **Reorder your source folders and source records to reflect the load order you want.**
   You can drag the objects around in a pane, just as you would in the Finder. You can delete objects that you don't want using the Delete key or the Clear command on the Edit menu.

8. **Before you save your project for the first time, see one of the following tasks on setting your project type as an application or a library.**

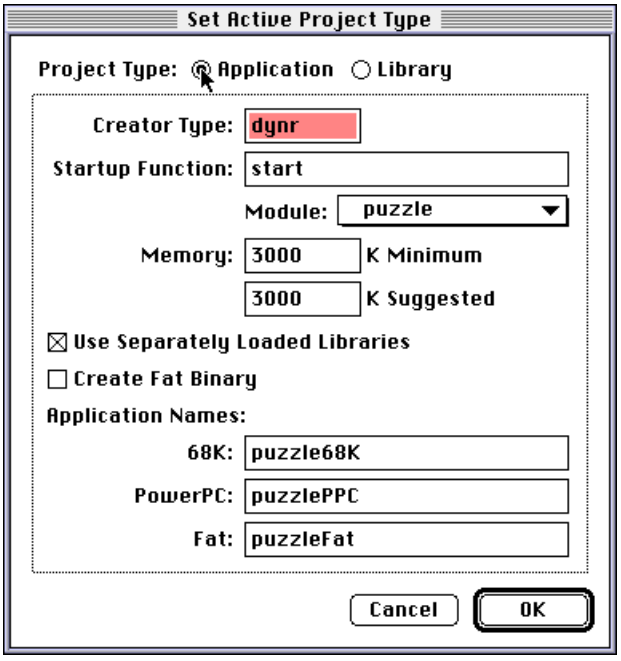
Setting the project type for an application

If you want your project to be an application, you must specify it as an application using the Set Project Type command on the Project menu. This information is used when you build your final standalone application from your project. If you have not yet written all the code needed to complete this dialog box, complete what you can and come back to this task when you have written the code.

If you want to use a project as a subproject, you should select Library in the "Project Type" field.

1. **Make your new project the active project, if it's not already.**
   You can use the Activate Project command on the Project menu.

2. **Choose Set Project Type from the Project menu.**
3. **Click Application in the "Project Type" field.**



4. **Enter a file creator in the "Creator Type" field.**
   The "Creator Type" field holds the creator ID you want your new application to have.
5. **Name your project's startup function in the "Startup Function" field.**
   This is a function you write to run your application.
6. **Select your startup function's module in the "Module" field.**
7. **Enter the memory requirements your application will have.**
8. **Check the box for separately loaded libraries, if your project requires them.**
   If this box is checked, the libraries used by the application are not bundled with the application file. They are left separate on disk. When the application is launched, it looks for the libraries it needs, first in its own folder, then in the extensions folder.

Separately loaded libraries save space because you only need one copy of the common libraries, dylan-framework and mac-toolbox.

Note that Dylan libraries are not shared in RAM, so selecting "Use separately loaded libraries" doesn't save RAM. Libraries are loaded when the application runs, if you check this box. They are also loaded whenever you issue a Launch Application Nub command in the active project.

The library of the application being written is always bundled with the application. Only the libraries of subprojects are kept separate.

9. **Click Create Fat Binary, if you wish.**
   A Fat binary includes both native 68K and native PowerPC code. It runs efficiently on both architectures, but is larger than an application created for only one architecture. The Fat binary is created when you use the Create Application command to build your standalone application.
10. **Edit the Application Names fields, if you wish.**
    Change these fields to the name you want your final application to have, according to the architecture you build it for.
11. **Click OK.**

Setting the project type for a library

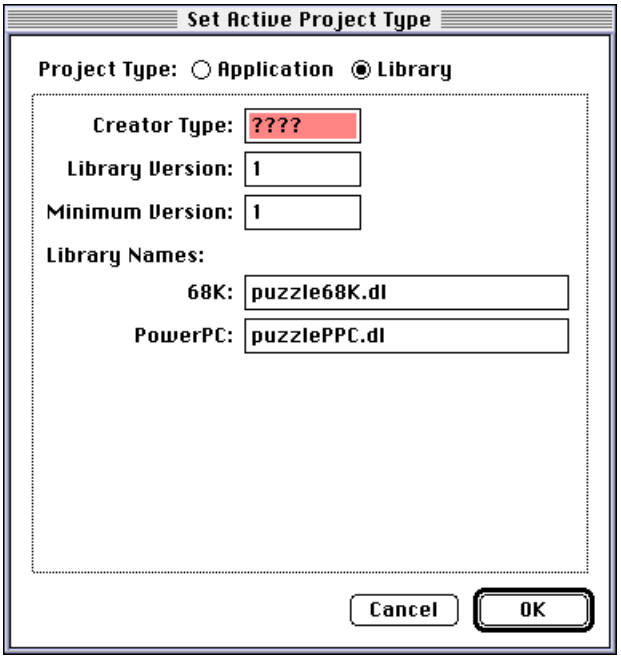
If your project will be a library, you must specify it as a library using the Set Project Type command on the Project menu. This information is used when you build your library from your project. If you have not yet written all the code needed to complete this dialog box, complete what you can and come back to this task when you have written the code.

Different fields appear in this dialog box when you select Library in the "Project Type" field rather than Application. That is because there is no startup function in a library and a library does not have memory requirements. Also, libraries do not vary according to whether they are separately loaded or not.

However, the versioning fields appear for Library. You should increment the version number of a library whenever you release a new version of a library. You also should increment the minimum version number whenever you make incompatible changes.

1. **Make your new project the active project, if it's not already.**
   You can use the Activate Project command on the Project menu.

2. **Choose Set Project Type from the Project menu.**
3. **Click Library in the "Project Type" field.**



4. **Give your project a version number and minimum version number.** You should increment the version number in the "Library Version" field whenever you release a new version of the library. You should increment the minimum version number in the "Minimum Version" field whenever you make incompatible changes, such as removing an export, changing the number of arguments of a function, changing the value of an exported constant, changing export or sealing information, changing the body of an accessible inlined method, changing the position of a slot in a primary class, or changing the behavior of some functions in a semantically non-upward-compatible way. In general, you should increment the minimum version number unless you very explicitly made only upward-compatible changes, such as adding an export, or adding optional arguments to a function.

5. **Select a file creator in the Creator Type field.**
   The Creator Type field holds the creator ID you want your new library to have.
6. **Edit the Library Names fields, if you wish.**
7. **Click OK.**

Saving a project

You save an entire project with the Save All command on the File menu. All the content in all open projects are saved. If you want to save only parts of a project, use the Save command.

The Save All command cannot be undone.

1. **Bring the project browser to the front, if it's not already there.**
2. **Choose Save All from the File menu.**
3. **If this is the first time you have saved it, give it a name and location.**
   The suffix ".π" is automatically appended to the name of your project when it is written to its folder in the Finder. Leave the development environment, if you wish, and check that its file was created in the proper folder.

Saving the objects in a pane

You can save the contents of a pane by clicking in the pane and then choosing the Save command from the File menu. The Save command cannot be undone.

You cannot save the contents of a single source record without saving the contents of every other source record in the pane with it.

1. **Click in the pane you want to save.**
2. **Choose Save from the File menu.**

Adding a subproject or resource file

You can add a subproject to a project with the Add to Project command on the File menu. Libraries are represented as subprojects in the development

environment. Use the Import command on the File menu to add a Dylan text file to a project. The framework is treated as a subproject by the development environment.

Not all projects can automatically be used as subprojects, only those whose project type has been set to "Library" using the Set Project Type command.

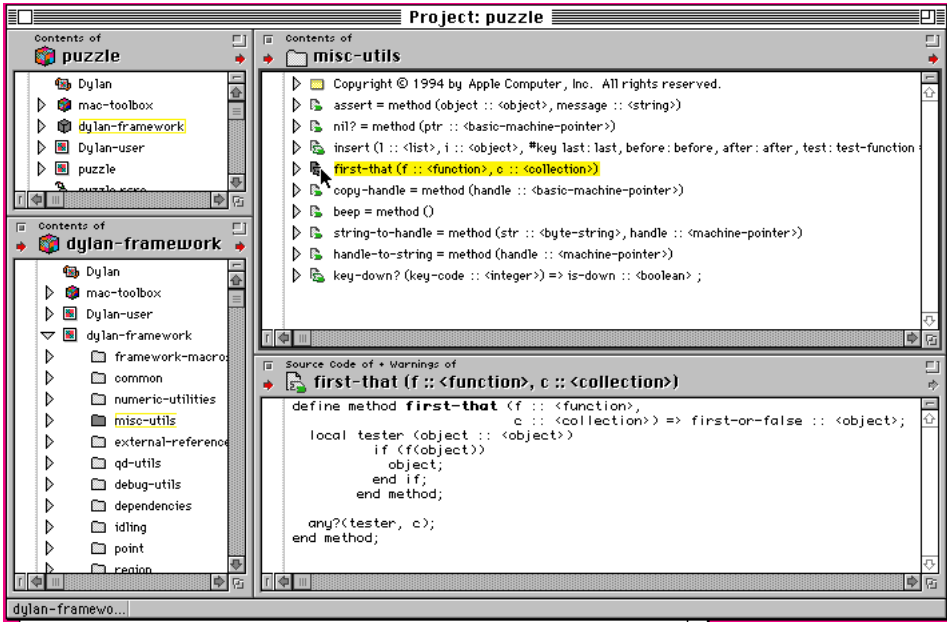
1. **Make active the project you want to add a subproject to.**
   You can use the Activate Project command on the Project menu.
2. **Make the root pane of the project browser active.**
3. **Choose Add to Project from the File menu.**
   If you are connected to the Application Nub, Apple Dylan alerts you that the Quit Application Nub command will be issued before you can proceed any further.
4. **Choose the file containing the subproject or resource file you want to add.**
   A dialog box allows you to find the file's location and choose the file to add. To add a project as a subproject, choose the project's file with the ".π" suffix. An alias to the subproject or resource file is also acceptable.
5. **Choose Update Project from the Project menu.**

Adding the framework

Although it is possible to write code without using the Apple Dylan framework, you probably will want to use it. The framework is a subproject that enables you to write less code. Source code for the framework is included in its project, so you can also use the framework to help you learn to write Apple Dylan code. See *Programming in Apple Dylan*, *Apple Dylan Extensions and Framework Reference*, and the *Dylan Reference Manual* in the Apple Dylan documentation set for more information on the framework.

1. **Make active the project you want to add the framework to.**
   You can use the Activate Project command on the Project menu.
2. **Choose Add to Project from the File menu.**
   If you are connected to the Application Nub, Apple Dylan alerts you that the Quit Application Nub command will be issued before you can proceed any further.

3. **Choose the framework's project file in the framework folder of Apple Dylan.**
   An alias to the framework is also acceptable.
4. **Choose Update Project from the Project menu.**
5. **Choose Launch Application Nub from the Project menu.**
   In this example, the framework has been added to this project and has been selected. The project browser has also had a fourth pane added to it.



Including C code

A C library can be included in a project by adding it using the Add to Project command on the File menu. You must add an entire C library, you cannot add individual snippets of C code.

If you want to investigate toolbox calls, select a call and choose the Look Up in Online Reference command on the Browse menu. The Macintosh

Programmer's Toolbox Assistant is launched, if you have purchased it and it is present on your system, and displays the reference entry for the call.

1. **Make active the project you want to add a C library to.**
   You can use the Activate Project command on the Project menu.
2. **Choose Add to Project from the File menu.**
   If you are connected to the Application Nub, Apple Dylan alerts you that the Quit Application Nub command will be issued before you can proceed any further.
3. **Choose the C library you want to add.**
   An alias to the C library is also acceptable.
4. **Choose Update Project from the Project menu.**
5. **Choose Launch Application Nub from the Project menu.**
6. **Select a toolbox call in a pane and choose the command Look Up in Online Reference from the Browse menu.**
   The Macintosh Programmer's Toolbox Assistant application is launched, if you have purchased it and it is loaded on your system. The entry for the command is displayed, but you can then navigate to other entries, including those for Apple Dylan.

Compiling your project

Now that you have created a new project, you can work on compiling code. The steps in creating an application in Apple Dylan are similar to what you are used to: you write code, compile it, debug it, build your application, and run it. However, the Apple Dylan development environment has a few features that might be new to you, such as incremental compilation. In the Apple Dylan development environment incremental compilation works on projects, modules, source folders, source records, and individual expressions in the code. You can even compile and execute code that has been commented out.

As you go through the following tasks you will notice that some menu items change to reflect the different compilation choices you make. For instance, if you select a source record in a pane and look at the Project menu, you see the command Compile Selection. If instead you open the source record, select just a portion of it, and then look at the Project menu, you see the command has

changed to Compile Region. If you have launched the Application Nub, the Compile Selection command becomes Compile and Download Selection on the menu.

Besides choosing to compile individual objects, you can choose to compile all the code changed since your last compilation with the Update Project command. Using the Recompile command, you can compile your entire project, regardless of when it was last changed. If you have added subprojects to your project, you can choose to update and compile them as well.

All these compilation commands compile the selected code and download it to the Application Nub, if it is tethered to the development environment. The Application Nub is the small (hence "nub") essence of an application that Apple Dylan provides. The Application Nub gives you a jump start on creating your own application in a project. As you write code in your project and compile it, it is being added into the Application Nub. When you build your final application, the Application Nub and your project, which have now merged, are bundled together and comprise your standalone application or library.

If you want to exclude some code from being compiled, you can use the Exclude Source Records command. All comment source records and commented out code are automatically excluded.

You can use the Apple Dylan Listener for investigating runtime objects. The Listener is a window with an interface that allows you to receive the results of your programming efforts or type in expressions to execute. The Listener reports all return values you get from executing code, even if the code is all in browsers, not in the Listener.

You can execute an individual expression by typing it in at the Listener prompt. This allows you to try out ideas without the overhead of creating a source record in a browser to hold the code. The code executed in the Listener is downloaded to the Application Nub.

To help you track each object in its development cycle, you can check its status indicators using the Status Indicators command on the Browse menu. The status indicators show whether an object has been compiled, saved, has generated any warnings, or is read-only. Status indicators propagate upward in the hierarchy of a project, if you set them in each pane of the project browser.

Launching the runtime

You do not need to launch the runtime (tether the development environment to the Application Nub or a standalone application) before compiling code from a browser. Your code will be compiled and stored in the compiler results database, then downloaded to the Application Nub the next time you launch it and tether to the development environment. However, when you want to use the Listener, you should launch the runtime because the results of any code you compile from there are lost the next time you tether the development environment to the Application Nub.

When you tether the Application Nub to the development environment, you can see that it is running by checking its status with the Application Nub Info command. You can also leave the development environment and check the list of running applications in the Finder. If you launched the Application Nub without also running an application or updating a project, and then switch to the Application Nub from the Finder list, you see that no windows or menus appear. That is because the Application Nub has no user interface. If you have opened a project and updated it or have an application running and switch to the Application Nub, you see the menu bar and user interface for the application replace the Apple Dylan menu bar. You can return to Apple Dylan by clicking in one of its windows.

You can launch the Application Nub automatically every time you open a project by setting the project's Listener preferences using the Preferences command.

1. **Open a sample project, if one is not already open.**
   You can also open one of your own projects.
2. **Choose Application Nub Info from the Project menu.**
   Notice the status is "Disconnected."
3. **Choose Launch Application Nub from the Project menu.**
   The notation "(Unconnected)" disappears from the Listener when the Application Nub is tethered to the development environment. You can now use the browsers or Listener to compile code, run the active project, or debug it. When you close the active project, the Application Nub automatically quits. When you reset the active projects to another open project, the Application Nub automatically quits and relaunches, downloading the compiled code from the new active project.

4. **Choose Application Nub Info from the Project menu.**
   Notice the status is "Connected."
5. **Go to the Finder list and choose Application Nub.**
   The Apple Dylan menu bar disappears if you opened a complete project that includes a menu bar, such as the samples. The menu bar won't disappear if you open a new, empty project.
6. **Click on an Apple Dylan window.**
   The Apple Dylan menu bar reappears and Apple Dylan is active.

Untethering from the runtime

You can quit the Application Nub using the Quit Application Nub command on the Project menu. If an application is running, you can also choose the Quit Application command to quit the Application Nub, which simultaneously quits your application.

If some problem occurs while programming and the Quit Application Nub command does not quit the Application Nub, you can leave Apple Dylan and use the program "Quit Application Nub" in the Application Nub folder.

1. **Bring the active project to the front.**
2. **Choose Quit Application Nub from the Project menu.**

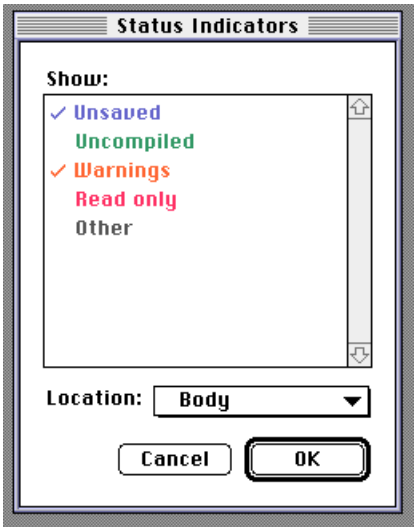
Checking code status

Status indicators show you the state of your unfinished code. For instance, you can tell which source records have not been saved by observing the Unsaved indicators in the status indicator bar in a pane.

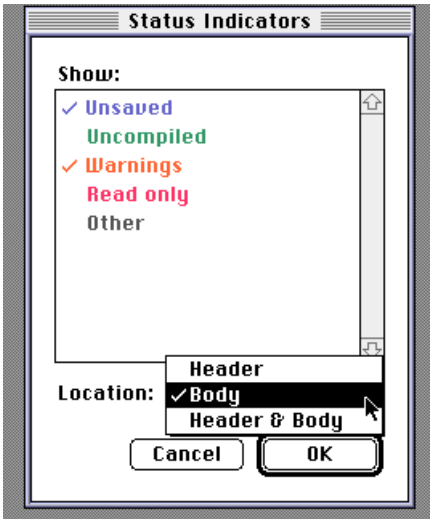
The status indicators can propagate upward in the source containers so you can see if a module or a source folder has an unsaved or uncompiled source record in it even if the source record is not visible. You set the status indicators for each pane by making the pane active and then opening the Status Indicators dialog box.

You can also choose not to see any status indicators or to see them in the header of a pane.

1. **Choose the Status Indicators command from the Browse menu.**
   You can also double-click in the vertical status indicator bar in a pane to open the Status Indicator dialog box. Drag it off the project browser so you can see both. Notice that unsaved source records and those with warnings are indicated by default in many panes.



2. **Choose Body as the location for your status indicators to be reported, if it's not chosen already.**
   You can choose to have the indicators appear in both the vertical status indicator bar in the pane (Body), in the pane's header (Header), or in both places. Click OK if you changed the location.



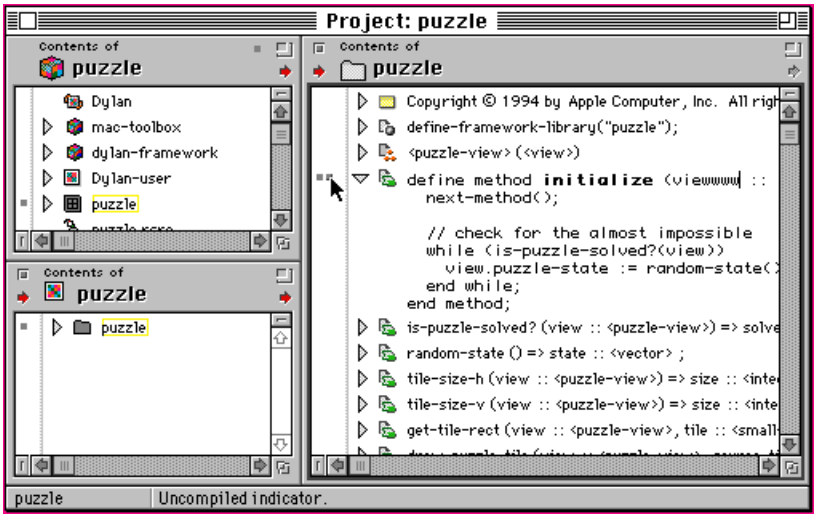
3. **Edit a source record in the project browser, but don't save it or compile it.**
   The Unsaved indicator appears. Notice that the Uncompiled indicator is not in the pane, even though you haven't compiled the source record yet. That is because that status indicator has not been chosen on the Status Indicators dialog box.

   You choose the type of status you want reported from the Show list. To choose a status indicator, click to its left until a check mark appears. If you choose all the indicators except Other, an indicator for each appears in your pane when that status applies.

4. **Choose Uncompiled from the Status Indicators dialog box.**
   The Uncompiled status indicator now appears in the row of indicators in the pane.

5. **Click on the second indicator in the pane and read its type in the prompt area at the bottom of the browser.**
   In the following figure, the cursor has been clicked on the Uncompiled indicator. The name of the indicator appears at the bottom of the browser in the prompt area. The first indicator in the row is the Unsaved indicator.

   The order of indicators is always the same, as is their placement in the vertical status indicator bar.



6. **Select only the Other status indicator in the Status Indicators dialog box, deselecting all those with check marks.**
   Notice that the three status indicators have been replaced by Other in the pane. Click it in the pane and the prompt area at the bottom of the browser shows you what indicators Other currently represents.

   The status indicator Other consolidates any of the unchecked indicators into a single indicator. By checking only the specific indicators you want to see, and also checking Other, you can see the specific indicators you want, but still be informed that other status changes have occurred.

Compiling a selection

You can compile a source record by selecting its icon and then choosing Compile Selection from the Project menu. You can also select one or more expressions, even commented out code, within a source record and then choose Compile Region to compile just the selected code. If you have launched the Application Nub, the commands on the menu are Compile and Download Selection or Compile and Download Region.

As you edit code, the source database grows. You should use the Compact Project command to reduce its size when it gets too big. As you compile and recompile your code using Compile Selection, the compiler results database

also grows; however, it cannot be compacted, so the only way to reduce its size is to use the Recompile command to build a new compiler results database from scratch.

You can choose to open the Warnings browser while you are compiling code. This can be helpful when you want to quickly examine objects that generate warnings. You can edit the code in the Warnings browser or in the original pane you were working in.

1. **Write or edit a source record in a pane so that it has a syntax error and won't compile.**
2. **Select the icon for the source record.**
3. **Choose Compile Selection from the Project menu.**
   You'll get a warning about the problem.
4. **Choose Undo from the Edit menu.**
5. **Select the revised code within the source record and choose Compile Region from the Project menu.**
   When you select code within a source record in a pane, either all of it or portions of it, the command on the menu is Compile Region.

   If you had opened the Warnings browser and edited the code in it, you could also compile it from there. The original source record in the original browser would also show the changes made in the Warnings browser.

Compiling all uncompiled code

You can compile any code changed in the active project and its subprojects since the last compilation using the Update Project command from the Project menu. The name of the project being updated follows "Update Project" on the menu. The Update Project command also downloads the compiled code to the runtime, if you are tethered to the development environment.

You can recompile all the code in a project and download it to the Application Nub by choosing the Recompile command from the Project menu. This is usually a lengthy process that you might not want to perform often, probably only when building your final standalone application or library, or in the event that the compiler results database grows too large.

When you mark a source record as included, it is included in all recompiles.

1. **Open the project you want to compile.**
2. **Choose Launch Application Nub, if you haven't already.**
3. **Include or exclude any source records, if needed.**
4. **Choose Update Project from the Project menu.**
   Notice that all uncompiled status indicators disappear from the project.

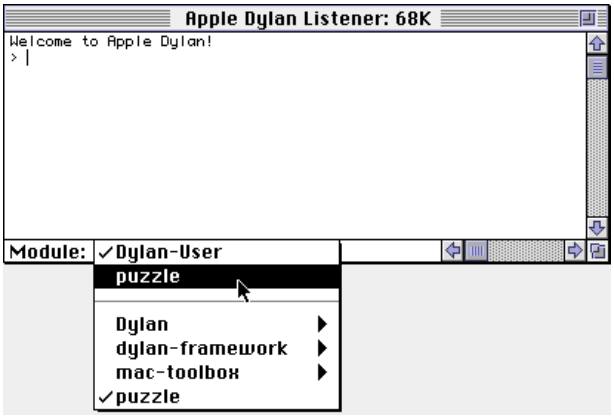
Compiling code from the Listener

You can compile code from the Listener rather than from a browser. You type or copy the code into the Listener at its prompt and hit the Enter key. You might do this to see if something yields the results you want before you save the code as part of your project or if some code in your project is not behaving as you want. You should launch the Application Nub before you use the Listener.

In the Listener you can execute an individual expression without creating a module, source folder, or source record to hold it. Any return values from the expression are printed into the Listener window. The Listener also prints the return values of expressions you execute anywhere in the development environment.

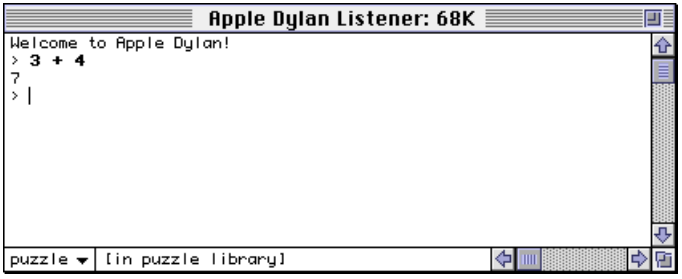
The expressions executed in the Listener are not saved when you quit the development environment as they are when they are in source records. However, you can print the contents of the Listener or copy code from it into a source record to save it.

1. **Make your project the active project, if it isn't, and launch the Application Nub.**
2. **Choose Apple Dylan Listener from the Windows menu to bring the Listener to the front.**
   If you can see the Listener window, you can also click in it to bring it to the front.
3. **Choose your project's module from the Module popup, which is in the lower-left corner of the Listener.**
   Be sure to choose the module that contains the code you want to compile, not the Dylan-user module. In the following example, the puzzle module has been selected.



4. **At the Listener prompt, type in an expression that yields a return value, such as 3 + 4, leaving the cursor somewhere in the expression or at its end.**
   You can also copy in code from a browser or another project.

5. **Press the Enter key.**
   The result is printed on the next line in the Listener. You can also choose Compile Expression from the Project menu to compile from the Listener. If you select text within an expression, the Compile Selection command appears on the menu and only the selected text is compiled.



   You can now enter other expressions at the next prompt or you can run any of them again without recopying them to the prompt by simply clicking the cursor in the expression and hitting the Enter key.

6. **Place the cursor into the expression again and hit the Enter key again.**
   You can recompile an expression by simply placing the cursor in it and hitting Enter, you don't have to copy the code to a new prompt.

7. **Place the cursor into the expression again and hit the Return key.**
   Any expression at a Listener prompt is printed at the next prompt by simply placing the cursor in the expression and hitting Return. You can then edit it before hitting the Enter key to compile the revised code.
8. **Edit the copied expression and hit the Enter key.**
   The edited expression is compiled with the return value printed below its prompt.

Excluding code from compilation

When you mark a source record, or part of a source record, as excluded, it is only compiled if you specifically select it before compilation. Otherwise, the excluded code will not be downloaded when the Application Nub is tethered to the development environment. You mark code as excluded by selecting it in a pane and choosing the Exclude Source Records command from the Project menu.

Excluded code is not built into your final application or library when you build it. Test code, unfinished code, unused alternate code, and example code should all be stored as excluded code.

1. **Select the icon for the source record or select the code within the source record to be excluded.**
2. **Choose Exclude Source Records from the Project menu.**

Including code in compilation

Source records are automatically marked as included when they are created. However, if you have previously excluded certain code, you might want to include it in future compilations. You mark code as included using the Include Source Records command from the Project menu.

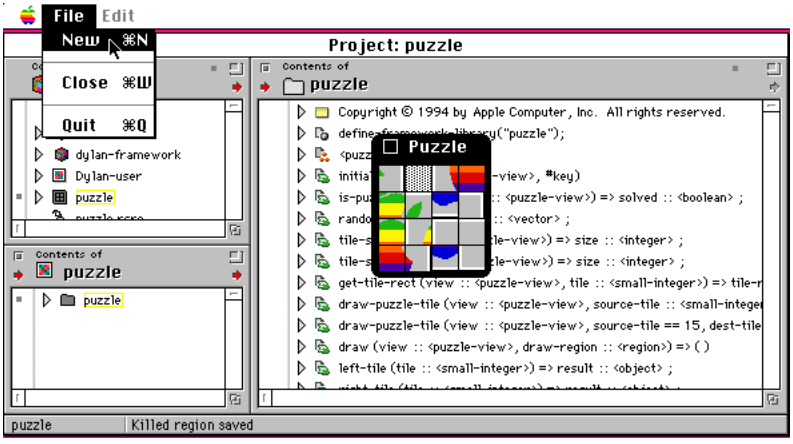
When you mark a source record as included, it will be included in all recompiles and updates. All included code is built into your final application or library when you build it.

1. **Select the icon for the source record or select the code within the source record to be included.**
2. **Choose Include Source Records from the Project menu.**

Running an application in Apple Dylan

You can run an application using the Run command on the Project menu. You can only run one application at a time from within Apple Dylan. Run launches the application after tethering it to the development environment and performing the Update Project command. The startup function for the application must be specified using the Set Project Type command before using the Run command.

1. **Open and make active a sample application, such as puzzle, or create your own.**
   To run your own application, you must write whatever basic code is needed, such as defining its library and module, writing its startup function, adding other subprojects as needed, etc., and successfully compile it. See the books *Programming in Apple Dylan*, *Apple Dylan Extensions and Framework Reference*, and the *Dylan Reference Manual*, which are included with Apple Dylan, for more information.
2. **Enter the startup function for the active project into the Set Project Type dialog box, if it's not already there.**
3. **Choose Run from the Project menu.**
   Notice that the menu bar for puzzle replaces the Apple Dylan menu bar and that the puzzle appears.



You can also run an application by typing its startup function into the Listener, if you choose the module it's in from the Listener's module popup.

To find the startup function for a sample application or other project not your own, use the Set Project Type command.

4. **Click on an Apple Dylan window.**
   Notice that the cursor is now the watch cursor. This is because puzzle is still running, although you can't currently see it. To quit the application from within Apple Dylan, choose the Quit Application command from the Project menu. This also quits the Application Nub. To quit just the application, you must get back to it by choosing Application Nub from the Finder list.
5. **Choose Application Nub from the Finder list.**
   You can now quit the application by choosing its Quit command from its File menu.

Note
You cannot use Apple Dylan again until you interrupt your running application. Use Quit within your application to do this.

Tethering to a running application

You can run an application from the Finder if a standalone version of it has been built. Then, you can tether it to the development environment if its compiler results database file was saved and is present.

To debug the standalone application, you run it, encounter a break, and then tether it to the development environment using the Tether to Application command on the Project menu. Your standalone application must be halted before you can tether it. You can cause it to halt by setting the event-check variable to be your own method for detecting a break request. For more information on event-check, see the book *Apple Dylan Extensions and Framework Reference*.

To change the standalone version, you must tether it, alter the code and then rebuild it as a new standalone application.

Standalone versions for all the sample applications appear in each sample's folder.

1. **Run an application from the Finder by double-clicking its executable file.**
2. **Halt the application.**

3. **Return to the development environment and open the standalone's project file using the Open command.**
4. **Make it the active project.**
5. **Choose Tether to Application from the Project menu.**
   You are now able to debug the application from within Apple Dylan.

You quit the application by choosing Quit Application in Apple Dylan or using its own Quit command.

Debugging a project

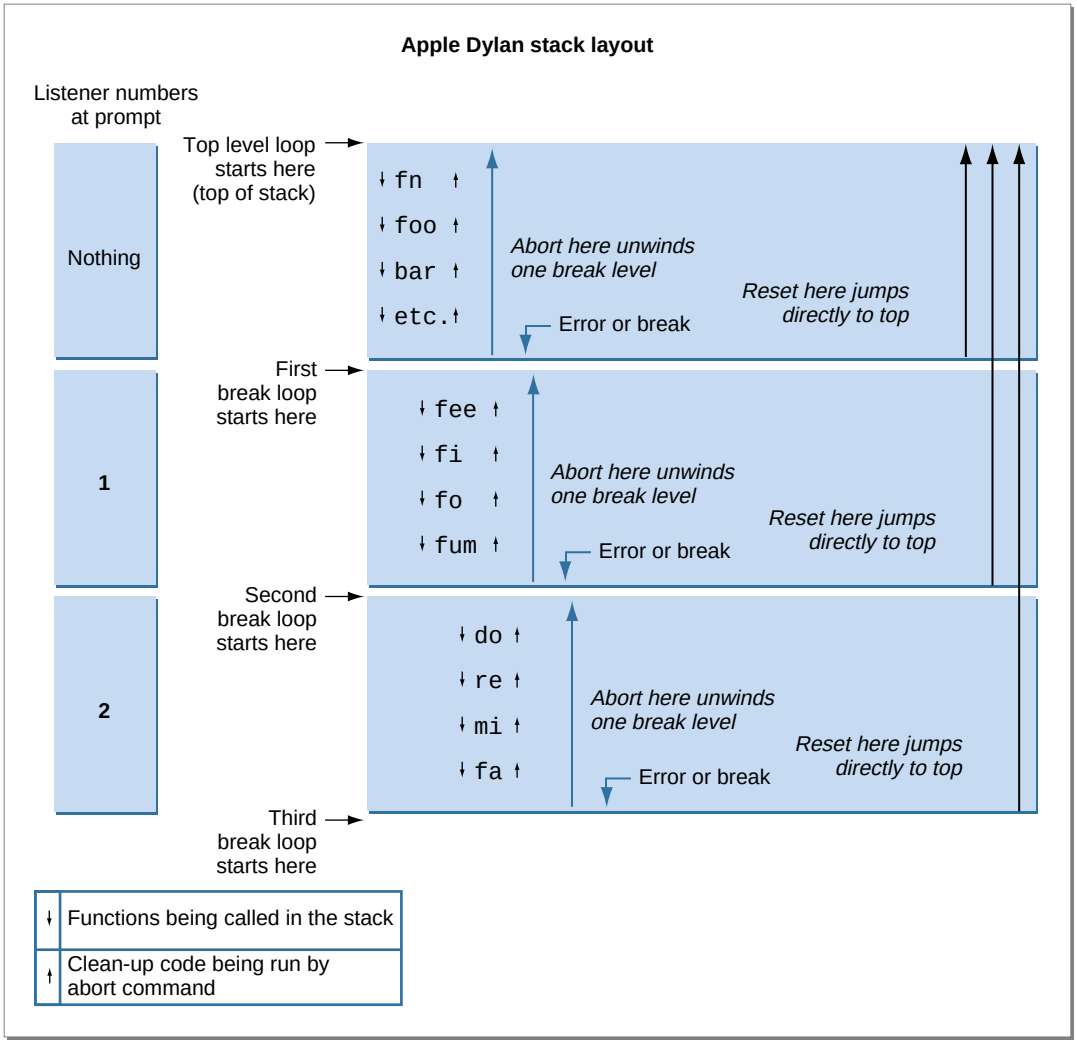
You can use the commands on the Debug menu to help you debug your application while it is running. Many of these commands work in concert with the Listener and inspector windows, allowing you to inspect runtime objects and work on performance problems.

You can run code from the Listener even when you have already made an error. This allows you to try out different ideas that might help you figure out how to fix the error. Any result printed to the Listener can be investigated further using the inspector window.

The commands that start with "Inspect" open inspector windows of some kind. There are several kinds of inspector windows, most of which are accessible from the Debug menu. Others you open by double-clicking on an object in another inspector window. For example, in the Listener when code results are returned, you can inspect those results in an inspector window using the Inspect Listener Results command. You could then double-click on an object in the inspector window and open another inspector window with that object as its basis. You can also select any expression in the Listener and choose Inspect Selection to inspect it in an inspector window.

The Reset Stack command discards the stack all the way to the top level without running any of the clean-up code. Because it does not run any clean-up code, this command is very risky; try the Abort command first. Severe side effects can result from not running the clean-up code. For example, a file can be left open with no way to close it or a data structure, such as the Dylan subproject itself, can be left in a corrupted state.

If your code has errors in it or if you issue several Break commands, the stack consists of as a series of break loops growing downward from the top level of the stack. The top level loop is the first loop executed and consists of function calls growing downward with each new command until an error or break is encountered. At the break point, you can either resolve the error, execute the cleanup code, or continue executing new commands, but any new commands executed are in another break loop, not the top level loop. Within the second loop another error or break might be encountered, in which case a third break loop is created. The Reset Stack command discards this entire set of loops and does not run the cleanup code, while the Abort command discards only the break loop you are in, executes the cleanup code, and returns you to the next break loop upward. The following figure shows the relative actions of Abort and Reset, as well as the Listener prompts that correspond to the break levels.

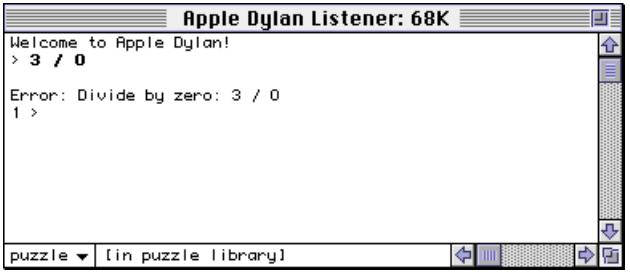


Inspecting the stack

When an error or a break is signalled, you can inspect the stack using the Inspect Stack command on the Debug menu.

You can execute code from the Listener while you are in a break loop. When you want to stop debugging and abort the operation, you can use the Abort command on the Debug menu.

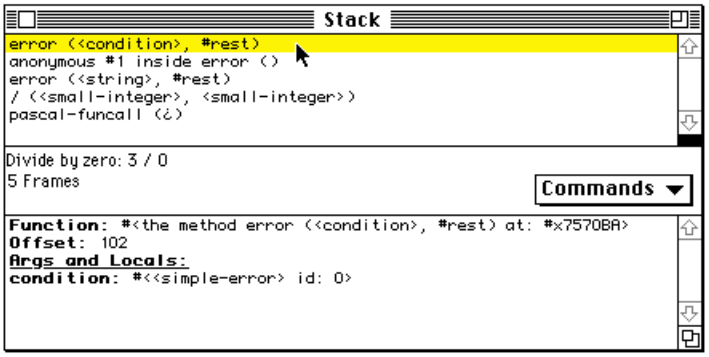
1. **Open a project, make it active, and choose Launch Application Nub, if you haven't already.**
2. **Choose the correct module from the Listener module popup.**
3. **Enter an incorrect expression, such as 3 divided by 0.**
   Notice that the Listener prompt has changed to the number 1 to indicate that you are in the first break level.



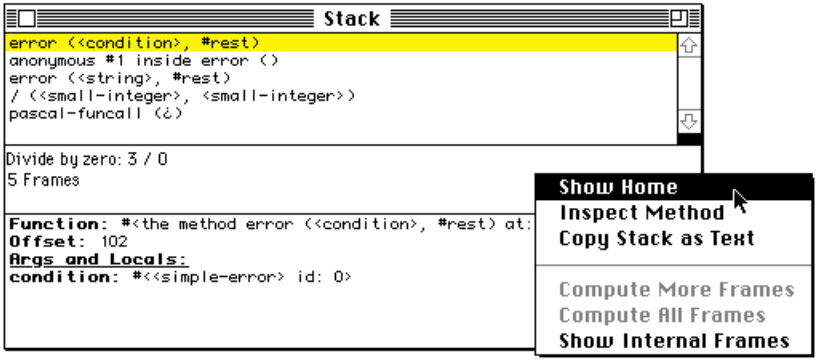
4. **Choose Inspect Stack from the Debug menu.**
   The Dylan Stack window appears, where you can examine the stack.
5. **Select a stack frame in the top pane.**
   When you select a stack frame in the top pane, the objects in it appear in the bottom pane. You can double-click on a frame in the top pane to open an editor for it where you can edit its source code.

   The bottom pane displays the active parameters and local variables for the frame selected in the top pane, as well as other information. You can inspect an object further by double-clicking on it in the bottom pane. This opens an inspector window on the object.

   In the following figure, the second stack frame in the top pane has been selected. In the bottom pane the function is named, its offset is given, and its arguments and local variables appear. You can change the type of information shown in the top pane by choosing one of the Dylan Stack commands on the Commands pulldown list in the center bar of the window. The commands include computing more frames, copying the stack as text, showing home (which displays the source code in a browser) and hiding internal frames.

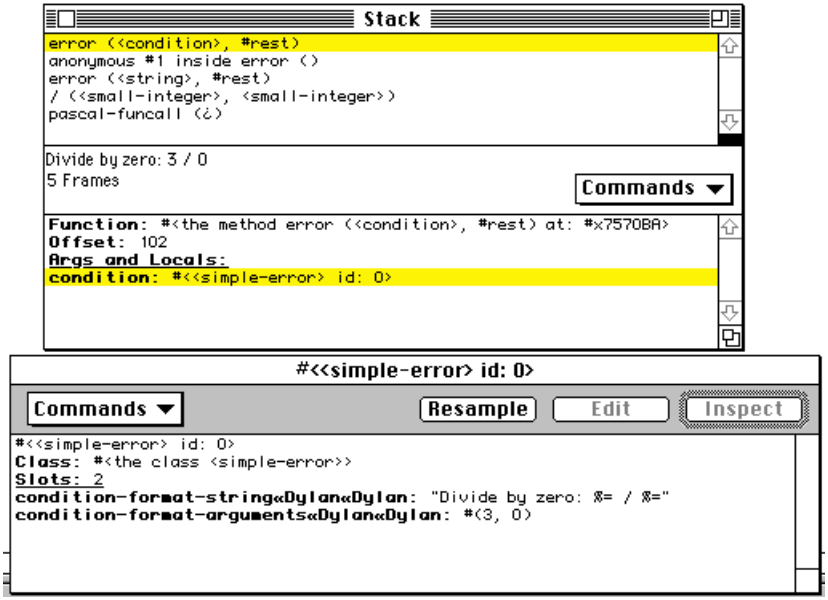


6. **Choose a command from the Commands pulldown list, if you want.**
   These commands affect the top pane.



7. **Double-click on an object in the lower pane to open an inspector window on it.**
   The following figure shows the first stack frame of the Dylan Stack window has been selected, which displays its contents in its bottom pane. An object in the lower pane has also been double-clicked, which displays an inspector window for that object.

   You can double-click any of the objects in the bottom pane to open an inspector window for it. If you have several inspector windows open at once, you can hold down the Option key while closing one and all of them close.



8. **Choose the Abort command from the Debug menu.**

Inspecting Listener results

You can inspect any results printed to the Listener using the Inspect Listener Result command. The following example uses the paint-app sample project, but you could use your new project if you want.

You can open another inspector window from any inspector window by double-clicking on an object in it. This can lead to having numerous inspector windows open at once. If you hold down the Option key while closing one inspector window, all of them close.

1. **Open the paint-app project, make it the active project, and issue Launch Application Nub to tether to the development environment.**
2. **Choose Update Project from the Project menu.**
   It is best that the project be synchronized to eliminate possible sources of confusion.

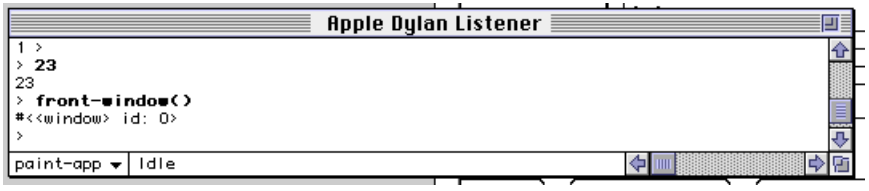
3. **Run the application.**
   You can either use the Run command from the Project menu or execute the startup function in the Listener. For a sample application, you can find the startup function using the Set Project Type command, which displays its name and location.

4. **Select the Application Nub icon from the Finder list to make it the front application and click on New on paint-app's File menu.**
   This opens a window in which you can create simple drawings. This step is necessary so the project will have some state that can be examined.

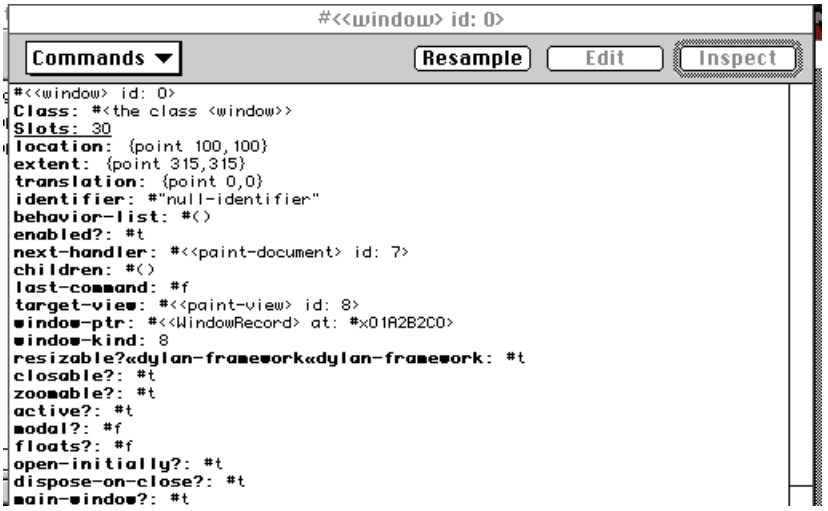
5. **Suspend the application.**
   You can do this by pressing Command-Option-Period while the application is running.

6. **Set the module you want to work with using the Listener's Module popup.**

7. **Return to the Listener and type in the name of a function you'd like to inspect.**
   In this case, 23 was entered. The Listener printed its value and then entered `front-window ()`, the name of one of the functions in paint-app.



8. **Choose Inspect Listener Result in the Debug menu.**
   An inspector window opens, showing all information about the current state of the variable `front-window`, including its name and class, plus all its slots and their contents.



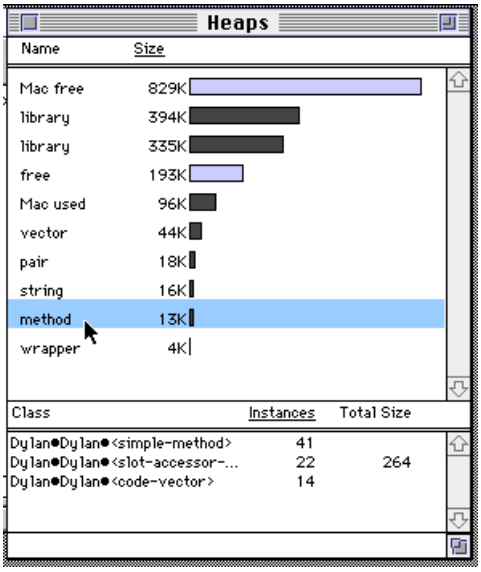
Inspecting heaps

You can inspect heaps with the Inspect Heaps command when you want to monitor how your application is using memory. The heaps shown in the Heaps window are a snapshot of all the heaps in the runtime, which are downloaded with a project or created by it. Macintosh heaps are not reflected in this window. Those heaps, such as for windows and menus, are created by the toolbox and are not monitored by the development environment.

There are several types of heaps in Apple Dylan, each for a different type of object. The heaps are calculated for you. When you select a heap in the top pane, its contents are displayed in the bottom pane. The top pane is a snapshot of the heaps taken when the Inspect Heaps command was issued. The bottom pane is a snapshot of the heap at the time you select the heap from the top pane. You can issue several Inspect Heaps commands to take snapshots at different times.

1. **Open a project, make it active, and choose Launch Application Nub, if you haven't already.**
2. **Choose Inspect Heaps from the Debug menu.**
   The Heaps inspector window opens.

3. **Select a heap you want more details on from the top pane.**
   Details about the heap appear at the bottom, including the classes in it, the number of instances of each class, and their total sizes. You can order the listing in the bottom pane by clicking the heading you want the list ordered by. In the following example, the list of classes is ordered by the number of instances, as is evident by Instances being underlined.



4. **Double-click on a heap in the top pane.**
   A window opens listing of all the classes in that heap.
5. **Double-click on a class from the list.**
   An inspector window on that class opens.
6. **In the Heaps window, double-click on a class in the bottom pane.**
   An inspector window on that class opens.

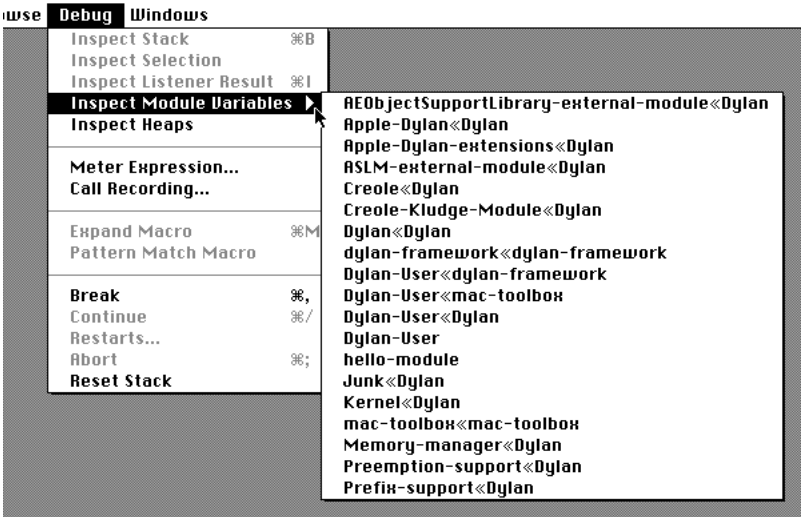
Inspecting modules

You inspect modules when you want to see all the module's variables in the Application Nub. You use the Inspect Module Variables command on the

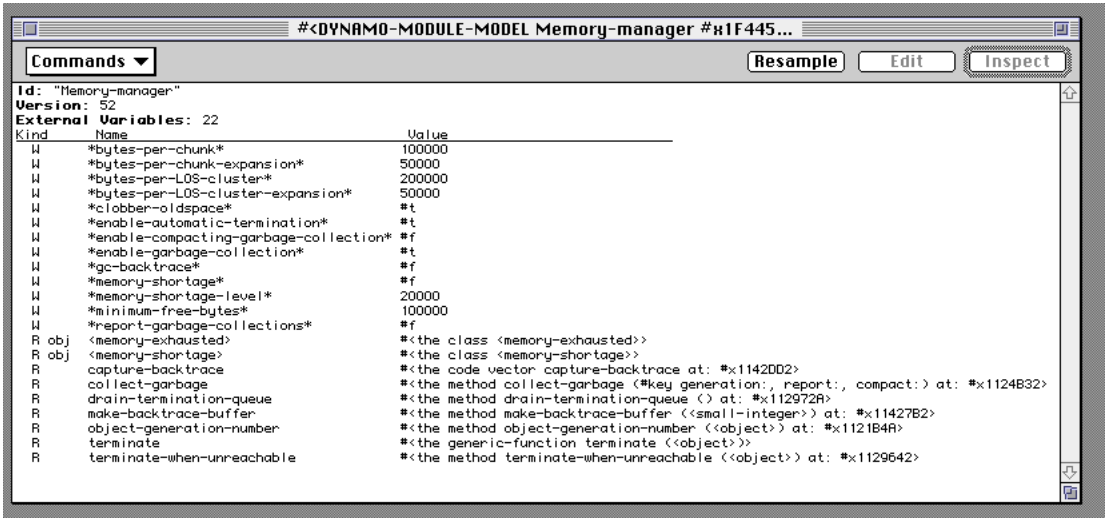
Debug menu and choose a module from the list, which opens an inspector window for it.

1. **Open a project, make it active, and choose Launch Application Nub, if you haven't already.**
2. **Choose Inspect Module Variables from the Debug menu.**
   From the list, choose the module whose variables you want to inspect.

   The following figure shows an example of the modules in a project's Application Nub.



The module inspector window displays the names, values, and kinds of variables that are defined in a module, as well as their read/write status. A menu item in the Commands popup menu toggles between "Show All Variables" and "Show Exported Variables."



The module inspector window shows only variables that are actually created in that module. It does not show variables imported from other modules, even if they are re-exported from the module being inspected. This is why, for instance, the inspector doesn't show any variables in the Dylan-user module.

Metering expressions

You can use the Meter Expression command on the Debug menu when you want to meter an expression in the runtime. When you type an expression into the top pane and run it, the generic functions called and classes allocated during the execution of the expression appear in the panes below. You can open an inspector window for an object in either of the lower panes by double-clicking on the object.

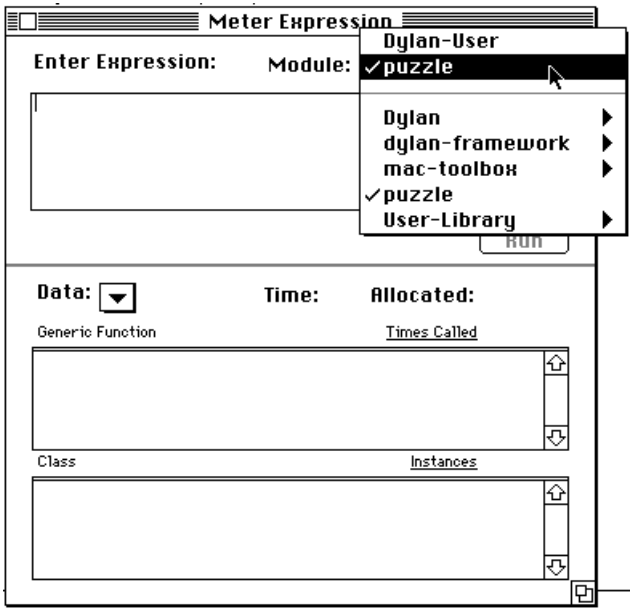
You must select the module the expression is in before metering it. You must be tethered to the development environment for Meter Expression to work.

The total time used and bytes allocated are also calculated in the Meter Expression window. The first time you call a function it sometimes runs more slowly than it will normally. If the generic function `Dylan.Dylan.finalize` appears on the list, this is the first time an instance of this class has been created. You should run the expression again for accurate timing.

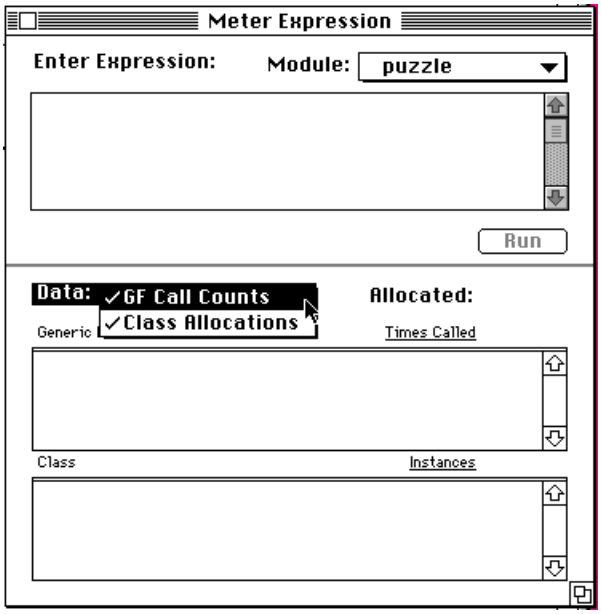
If the generic function `Dylan.Dylan.set-generic-function-dispatch` appears on the list, it indicates that this is the first time one of the generic functions has been called. Sometimes if you add a method to a generic function, it needs to be treated as if it is being called for the first time. You should run the expression again to get more accurate timing results.

Use the Call Recording command to trace or meter an individual function.

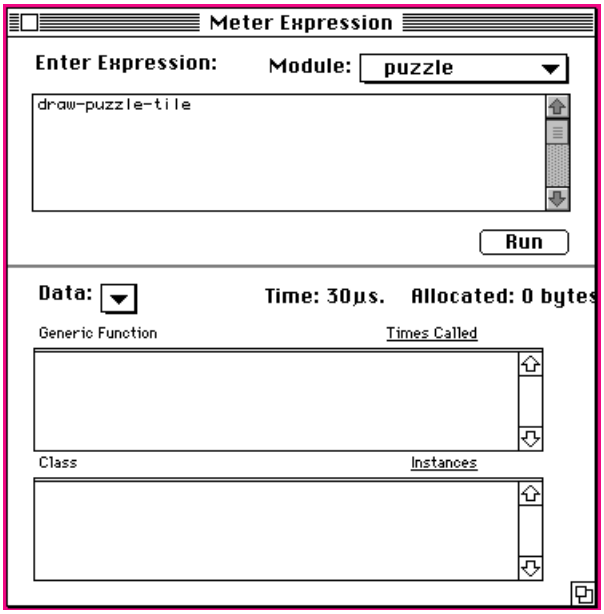
1. **Make your project active and launch the Application Nub, if you haven't already.**
2. **Choose Meter Expression from the Debug menu.**
3. **Choose the module the expression is in from the Module field.**



4. **Choose the type of information you want to see for the expression in the Data field.**
   Generic function calls that have been inlined are not displayed in the Generic Function pane. Some built-in classes, such as <list> and <byte-string>, are not displayed in the Class pane but are counted in the Allocated bytes field.



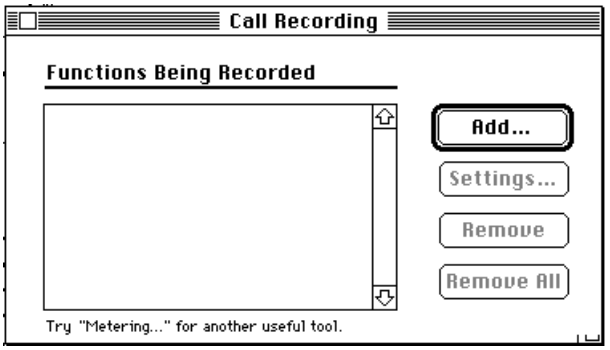
5. **Type in the expression and choose Run.**
   You can also copy and paste an expression into the top pane. The metering information is printed into the Time and Allocated fields and in the panes at the bottom of the window. You can order the listing in the bottom panes by clicking the heading you want the list ordered by.



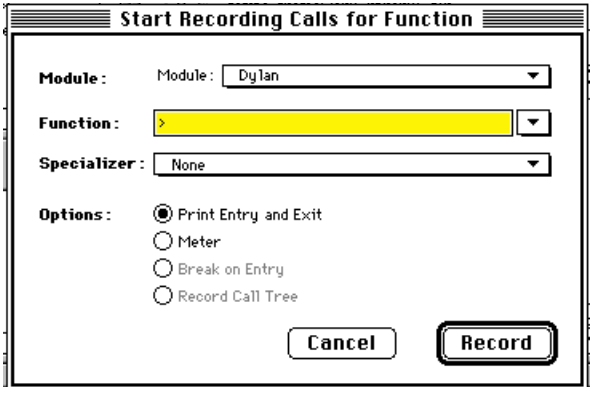
Monitoring an individual function

You can trace the behavior of an individual function, such as a method or generic function, in the runtime using the Call Recording command on the Debug menu. The function is printed to the Listener on entry. You must be tethered to the development environment for Call Recording to work.

1. **Make your project active and launch the Application Nub, if you haven't already.**
2. **Choose Call Recording from the Debug menu.**
   The Call Recording window opens.

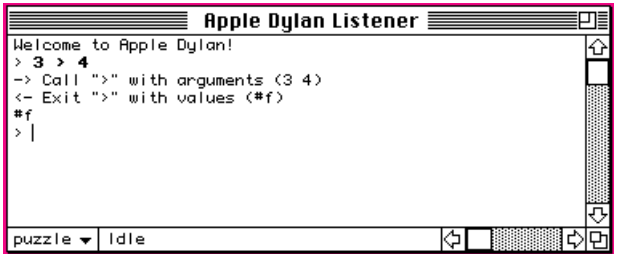


3. **Choose Add to select the function to record.**
   The Start Recording Calls for Function window opens.
4. **Choose the module the function is in from the Module field.**
5. **Choose the function to record from the Function field.**
6. **Choose the specializer to trace from the Specializer field.**
7. **Choose whether you want to print the function on entry and then exit, or to meter the function.**
   In the following example, the greater than symbol in the Dylan subproject has been selected.



8. **Click Record when you have made your choices.**
   The recorded function appears in the Call Recording window. You can remove functions from the list with the Remove and Remove All buttons. Select the functions you want removed and click the appropriate button.

9. **The results are printed to the Listener.**



Creating a user interface

You can create a user interface for your application through the Apple Dylan user-interface builder. Although it is possible to create a user interface without using the user-interface builder, this is not the suggested practice. The user-interface builder is a graphical utility that helps you visualize your user interface before writing any code for it, thus saving time as you refine your design. See the book *Creating a User Interface in Apple Dylan* for more information.

To activate the user-interface builder, use the Load UI Builder command on the Project menu. This loads the builder's library into the Application Nub and makes the command Show Interface Builder appear on the Apple menu. Make sure you have have launched the Application Nub before using the Load UI Builder command.

You then choose Show Interface Builder from the Apple menu to launch the builder. This does not quit the development environment, but simply hides it while you run the builder.

As you create your user interface using the builder, the high-level elements of your interface, such as windows and menus, are saved by the builder to a project file. This project file contains the user interface elements as resources, although it is not a typical resource file. You then add the builder project file to the development environment project it belongs to.

If you want to use a custom icon for your application, you must create the icon using an icon editor outside of Apple Dylan. You then simply add the resource file it is in to your project. If you don't use a custom icon, the default Macintosh icon for applications is used.

Adding the Apple Dylan interface builder

You run the builder from the development environment project you want to create a user interface for. Note that you can run the builder as a standalone application from the Finder by double-clicking its standalone version. However, some of the capabilities of the builder are not available if you run the standalone version.

1. **Make active the project you want to add a user interface to.**
2. **Launch the Application Nub, if it isn't running already.**
   You should also add the framework to your project, if you haven't already.
3. **Choose Load UI Builder from the Project menu.**
4. **Choose Show Interface Builder from the Apple menu.**
   You can return to the development environment, without quitting the builder, by choosing the Hide Interface Builder command from the Apple menu. In this way you can quickly move back and forth between the development environment and the builder without quitting either. It's a good idea to save your builder project before hiding it. When you do quit

the builder, the Application Nub in the development environment and your running application are also quit.

5. **Choose Launch Application Nub from the Project menu.**
6. **Launch the user-interface builder using the Show Interface Builder command from the Apple menu.**
   See the book *Creating a User Interface in Apple Dylan* for information on using the interface builder.

Adding the new user interface

When you have run the user-interface builder and created the user interface for your application, your user interface's data is saved in a project file. You can then add this project file to the development environment project it belongs with. The project file with your user interface in it appears in your development environment project as a resource file.

You must also write code to attach the user interface elements to your project as described in the book *Creating a User Interface in Apple Dylan*.

You can then run your application to see its user interface.

1. **Create the user interface and write the code needed to attach the user interface elements to your project.**
2. **Choose Add to Project from the Project menu in the development environment.**
   If you are connected to the Application Nub, Apple Dylan alerts you that the Quit Application Nub command will be issued before you can proceed any further.
3. **Locate and choose the project file containing the user interface.**
   The file will have the name you gave it, plus the suffix "rsrc".
4. **Choose Launch Application Nub from the Project menu.**
5. **Choose Update Project from the Project menu.**
6. **Run your application using the Run command on the Project menu.**

Sharing your user interface

You can share your user interface with others by sending them your builder project file. Then, they can add it to their projects and can also alter it by running the user-interface builder.

1. **In the user-interface builder, save your user interface.**
2. **In the Finder, send the project file to someone.**
   They can open your project file with their version of Apple Dylan, add it to a project, and alter it in the builder.

Building your application or library

When you have written your code, compiled, and debugged it, you can build your final standalone application or library. To build an application or library from your project, you choose the Create Application command or Create Library command on the Project menu.

Be sure to recheck the accuracy of the project settings with the Set Project Type command before building your final application or library.

You should probably recompile all the code in your project before building your final standalone application or library by choosing the Recompile command from the Project menu. This can be a lengthy process.

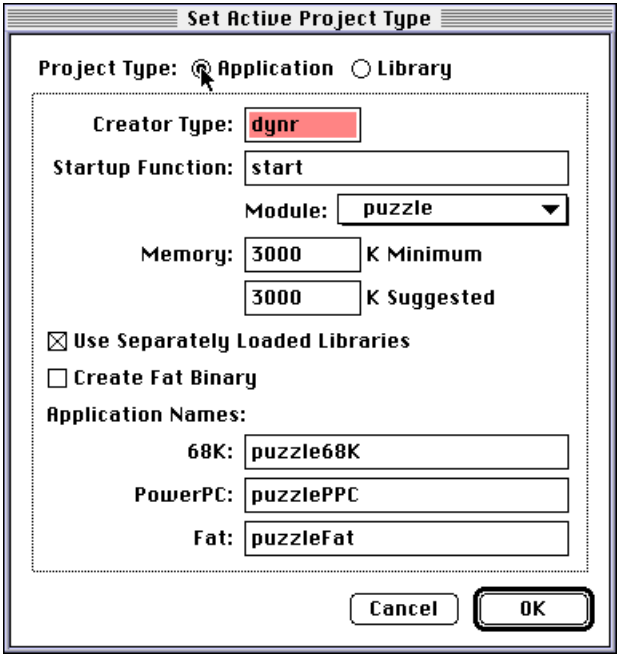
Building your standalone application

You build your final standalone application from your project using the Create Application command on the Project menu. The Create Application command appears on the Project menu when you set the project type as an application using the Set Project Type command.

Before you build a standalone application, you should quit the Application Nub; you will be prompted to do so in the Set Project Type dialog box. You must also check that the information about your project is all correct in the Set Project Type dialog box.

If you need to debug the standalone application in the future, you can tether to it and return to the use of all the facilities of the development environment. To do this, you must have retained the project's compiler results database file.

1. **Make your project the active project, if it's not.**
   The Activate Project command from the Project menu allows you to designate the active project.
2. **Choose the Set Project Type command from the Project menu.**
   You should have set these values already, but you should make sure the fields are all correct.



3. **Click OK.**
4. **Choose the Create Application command from the Project menu.**
5. **Click the destination platform for your application.**

6. **Keep the database files for your project.**
   To alter your standalone application in the future, you can tether to it, edit the code in the development environment, and rebuild it as a new application only if you have at least kept the compiler results database file.
7. **Leave Apple Dylan to make sure the file was created and runs.**

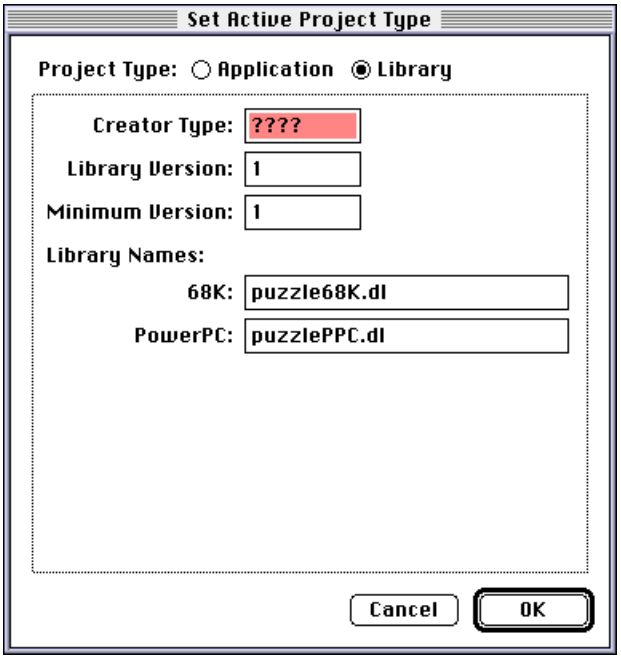
Building a library

You build your final library from your project using the Create Library command on the Project menu. The Create Library command appears on the Project menu when you set the project type as a library using the Set Project Type command.

Before you build a standalone library, you should quit the Application Nub; you will be prompted to do so in the Set Project Type dialog box. You must also check that the information about your project is all correct in the Set Project Type dialog box.

If you need to debug the library in the future, you can tether to it and return to the use of all the facilities of the development environment. To do this, you must have retained the project's compiler results database file.

1. **Make your project the active project, if it's not.**
   The Activate Project command from the Project menu allows you to designate the active project.
2. **Choose the Set Project Type command from the Project menu.**
   You should have set these values already, but you should make sure the fields are all correct.



3. **Click OK.**
4. **Choose the Create Library command from the Project menu.**
5. **Keep the database files for your project.**
   To alter your library in the future, you can tether to it, edit the code in the development environment, and rebuild it as a new application only if you have at least kept the compiler results database file.
6. **Leave Apple Dylan to make sure the file was created.**

Sharing code

You can share code containers, either source folders or modules, with other programmers by exporting them as text files, but you cannot share source records individually. You can save your personal browsers and send other

programmers the browser file. You can also share a user interface by sending others its resource file, which was generated by the user interface builder.

Remember, you should copy the files you want to share; moving or deleting Apple Dylan files while not in Apple Dylan can have bad consequences, as does renaming any folders. The development environment will not be able to find the object and report it as missing. Missing files need to be found and identified when the development environment is running or they will just be ignored during compilations. When they have been identified, the project must be recompiled and saved. While the development environment is not running, you can move the entire Apple Dylan folder or a project's entire folder and it will be OK.

Sharing projects

You can share an entire project with someone by sending them a project's folder. You can send them a subproject by sending its project folder. You don't have to send them the sources when sending them the project. To use the project or subproject, they add it to the project they want it in.

1. **Save a project if you haven't.**
   You send the file it's saved into.
2. **Leave the development environment.**
3. **Send the project's file.**
   Send the entire folder, including the project's databases, if you want the other person to be able to alter your project.

Sharing code by exporting

You can share source code by exporting it. You export a container object from the development environment with the Export command on the File menu. Everything within the container is exported automatically. You cannot export individual source records. The exported code is written to a file, called a "Dylan text file", with the suffix ".dylan".

The Dylan text file has a header on it to identify its contents. These files can be mailed electronically or edited outside of the development environment. An exported object can be brought back into the development environment by importing the file it is in.

However, not every object can be exported and imported without changes.

- A single source record with more than one top-level form or comment is broken into an individual source record for each top level form or comment when exported.
- In addition, if any code within a source folder has the incorrect number of begins or ends, or is any other way seriously malformed, the source folder can be exported, but the resulting Dylan text file cannot be imported.

Importing or exporting has no effect on objects marked for exclusion or inclusion.

1. **Select the object you want to export.**
2. **Choose Export from the File menu.**
   Dylan text files have the suffix ".dylan".
3. **Choose the location for the Dylan text file.**
   The following figure shows a portion of a Dylan text file.

```
language: infix-dylan
module: text-app


define class <text-app-library> (<library>)
end class;

define constant $text-app-library = make(<text-app-library>, name: "text-app.rsrc")

define class <text-app-behavior> (<behavior>)
end class;

define method behavior-setup-menus (behavior :: <text-app-behavior>,
                                    next-behaviors :: <list>,
                                    main-handler :: <main-handler>) => ()
 ignore(next-behaviors, main-handler);
 next-method();

 enable-item(#"new");
end method;

define method behavior-event (behavior :: <text-app-behavior>,
                              next-behaviors :: <list>,
                              main-handler :: <main-handler>,
                              event :: <menu-event>,
                              id == #"new") => ()
 ignore(behavior, next-behaviors, main-handler, event, id);

 open(make(<text-document>));
end method;

define method behavior-event (behavior :: <text-app-behavior>,
                              next-behaviors :: <list>,
                              main-handler :: <main-handler>,
                              event :: <open-application-event>,
                              id :: <object>) => ()
 ignore(behavior, next-behaviors, main-handler, event, id);

 open(make(<text-document>));
end method;

define constant $file-menu-id = 128;

define constant $edit-menu-id = 129;

define constant $font-menu-id = 131;
```

Retrieving code by importing

You retrieve the contents of a Dylan text file to the development environment by importing it. You import Dylan text files with the Import command on the File menu.

You create Dylan text files by exporting objects from Dylan. Exported files can be edited in other word processors and mailed electronically.

However, not every object can be exported and imported without changes.

- A single source record with more than one top-level form or comment is broken into an individual source record for each top level form or comment when exported.
- In addition, if any code within a source folder has the incorrect number of begins or ends, or is any other way seriously malformed, the source folder can be exported, but the resulting Dylan text file cannot be imported.

Importing or exporting has no effect on objects marked for exclusion or inclusion.

1. **Choose Import from the File menu.**
2. **Find the Dylan text file you want to import.**
   Dylan text files have the suffix ".dylan".
3. **Choose Import from the dialog box.**

Apple Dylan Reference

This is the reference chapter for the Apple Dylan development environment.

It includes:

- List of command shortcuts
- Alphabetical list of all commands

Key Command Shortcuts

Not all commands have keyboard equivalent shortcuts. Those that do are listed below and also appear in the full descriptions of the commands under the subheading "Key shortcut".

File Menu

- New Source Record—**Command-N**
- Open—**Command-O**
- Close—**Command-W**
- Save— **Command-S**
- Print— **Command-P**
- Quit—**Command-Q**

Edit Menu

- Undo (Clear, Typing)—**Command-Z**
- Redo—**Command-Z**

- Cut—**Command-X**
- Copy—**Command-C**
- Paste—**Command-V**
- Select all—**Command-A**
- Copy Special
  - Argument List—**Command-'**
  - Define Method Template—**Command-T**
  - Class Template—**Command-T**
- Insert Special
  - Method Template—**Command-Option-T**
  - Class Template—**Command-Option-T**
  - Argument List—**Command-Option-'**

Text Menu

- Style
  - Plain—**Command-Shift-T**
  - Bold—**Command-Shift-B**
  - Italic—**Command-Shift-I**
  - Underline— **Command-Shift-U**
  - Outline—**Command-Shift-O**
  - Shadow—**Command-Shift-S**
  - Condense—**Command-Shift-Option-C**
  - Extend—**Command-Shift-E**
- Find and Replace—**Command-F**
- Find Again—**Command-G**
- Find Selection—**Command-H**
- Replace and Find—**Command-J**

Project Menu

- Run—**Command-R**

- Launch Application Nub—**Command-K**
- Compile Region, Expression, Selection—**Command-E**
- Update Project—**Command-U**

Browse Menu

- Aspect—**Command-Option-click** on object
- Show Home—**Command-Y**
- Look Up in Online Reference—**Command-=**

Debug Menu

- Inspect Stack—**Command-B**
- Inspect Listener Result—**Command-I**
- Expand Macro—**Command-M**
- Expand Macro one level—**Command-Option-M**
- Break—**Command-,**
- Continue—**Command-/**
- Abort—**Command-;**

Command Reference

The following list is alphabetized with the commands on the left margin. The right margin shows the menu the command is on. Menus are altered dynamically. Not all commands are available in all contexts. For instance, Launch Application Nub and Quit Application Nub replace each other on the Project menu.

Abort — Debug

Discards the stack in a break level loop to the next break level upward in the stack after executing the clean-up code. Because it runs clean-up code, Abort is usually preferable to Reset Stack, which does not.

See "Debugging a project" on page 146 for a comparison of using Abort and Reset Stack.

Related command: Reset Stack.

Key shortcut: **Command-;**

About Apple Dylan … — Apple

Displays information about Apple Dylan, including copyrights, credits, memory usage, and version information.

Activate Project — Project

Makes the selected open project into the active project.

Activate Project does the following:

- Makes the currently active project inactive, leaving its browsers, databases, and project files open.
- Untethers the Application Nub containing the currently active project from the development environment, closing the compiler results database and removing it from memory.
- Makes the selected project active, connecting to its compiler results database, making it available for browsing.
- If Update on Open is set (in Preferences) the project is updated.
- If Connect on Open is set (in Preferences) the project is loaded into the Application Nub.

See "Apple Dylan User Model" on page 95.

Add to Project — File

Adds a subproject, C header file, text file, library, or resource file to a project. Aliases are supported. Before you can use the added subproject or file, you must use the Update Project command. This command disconnects you from the Application Nub if you are connected to it.

You can drag and drop objects between projects to achieve the same end.

Use Add to Project to include the framework, mac-toolbox, or the user-interface builder in a project.

All Methods of (Aspect) — Browse

Operates on a selected class, displaying the inherited and direct methods of the class in a pane.

Related command: Direct Methods of (Aspect).

All Methods of (Aspect) is commonly used to customize a browser pane displaying that aspect of a class. You can also select a single class in a pane and display all its methods inline.

See “Aspect” on page 183 and “Customizing browsers” on page 58 for more information.

All Slots of (Aspect) — Browse

Operates on a selected class, displaying the inherited and direct slots for the class in a pane.

Related command: Direct Slots of (Aspect).

All Slots of (Aspect) is commonly used to customize a browser pane displaying that aspect of a class. You can also select a single class in a pane and display all its slots inline.

See “Aspect” on page 183 and “Customizing browsers” on page 58 for more information.

All Subclasses of (Aspect) — Browse

Operates on a selected class. Displays inline the inherited and direct subclasses for the class in a pane.

Related command: Direct Subclass of (Aspect).

All Subclasses of (Aspect) is commonly used to customize a browser pane displaying that aspect of a class. You can also select a single class in a pane and display all its subclasses inline. See "Aspect" on page 183 and "Customizing browsers" on page 58 for more information.

All Superclasses of (Aspect) — Browse

Operates on a selected class. Displays inline the inherited and direct superclasses for the class.

Related command: Direct Superclasses of (Aspect).

All Superclasses of (Aspect) is commonly used to add a custom browser pane displaying that aspect of a class. You can also select a single class in a pane and display all its superclasses inline.

See "Aspect" on page 183 and "Customizing browsers" on page 58 for more information.

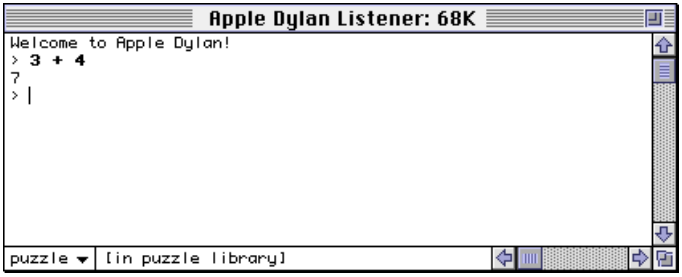
Apple Dylan Listener — Windows

Selects the Apple Dylan Listener. Once the development environment has been tethered to the Application Nub, the Apple Dylan Listener window opens and stays open unless you untether from the Application Nub, when the name changes to "Apple Dylan Listener (Unconnected) :68K". The type of target machine chosen using the Target Architecture command is also shown in the Listener's title.

The Apple Dylan Listener allows you to execute code in a running application. The Listener prints all returned values executed in the development environment, even when the expression was not executed in the Listener.

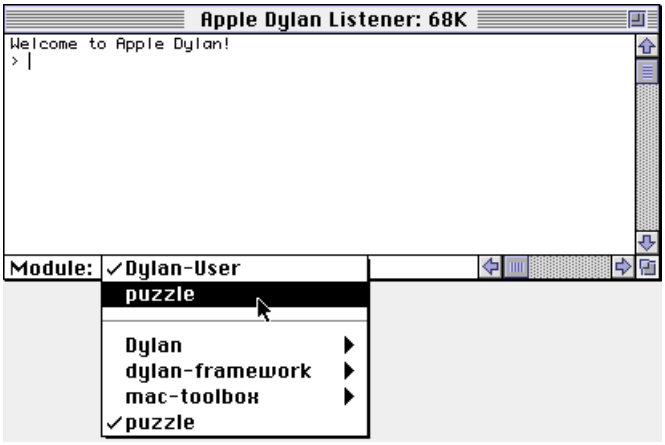
Use the Listener by typing in Dylan language constructs at the prompt, and then pressing the Enter or Return key. When the module the code resides in

and the module the Listener is set to are different, the module appears at the prompt. This is also true if the libraries differ.



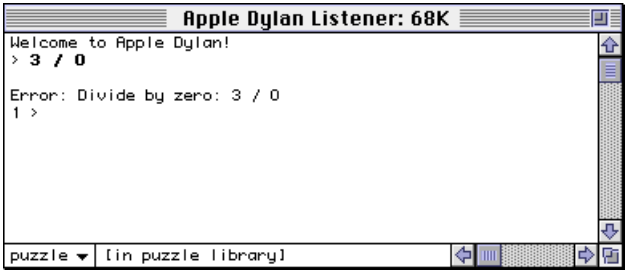
Note
Before you type an expression into the Listener, choose the proper module from the popup list in the lower left corner of the Listener window.

The current library is shown at the bottom of the Listener window to the right of the Module popup. When you hold down the mouse on the Module popup, as shown in the following figure, the modules in the current library appear in the list above the separator line. When you choose one of these modules, it becomes the module the Listener uses. The current module and library are check marked in the popup. Below the separator line are other libraries you can choose from. If you choose one of the libraries below the separator line, it becomes the current library and, when you release the mouse, is shown at the bottom of the Listener window. That library's modules appear above the separator line the next time you access the Module popup.



You can also select expressions in the Listener and choose commands from the Apple Dylan menus. The expression is compiled, linked to the Application Nub, and executed immediately. The Listener responds by printing its result or return values on the line following the prompt. The Listener does not save any code; you must copy the code into a source record to save it.

You can only use the Listener when it is connected to the Application Nub and your application is running. You can, however, execute code from the Listener while you are in a break loop, allowing you to investigate various results. When in a break loop, the Listener's prompt changes to the number of the break level on the stack, starting with the first break loop as number one.



The Listener is also useful for quickly testing expressions without creating all the scaffolding to hold them. In addition, the Listener provides access to parts of your application that you cannot access any other way, such as data stored in a database or off a hash table.

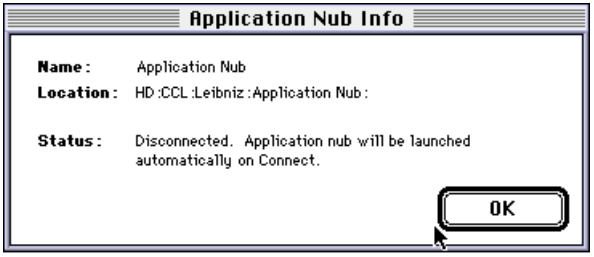
Related command: Inspect Listener Result.

Note
To suspend a running application, make it the front application and type command-option-period. When the application has stopped you can issue Quit Application Nub.

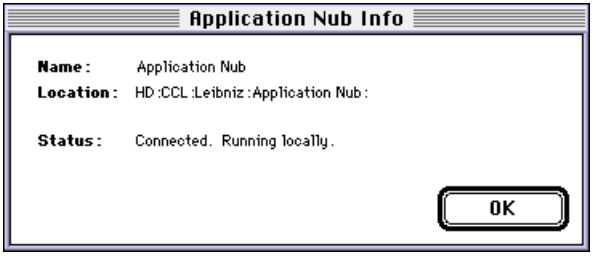
Application Nub Info — Project

Displays information on whether the active project is loaded into the Application Nub.

The following figure shows the status of an untethered Application Nub.



The following figure shows the status of an Application Nub that is tethered and running locally. If the Application Nub were on a second machine, its status would be Running remotely.



Argument List (Copy Special) Edit

Copies the prototype argument list for the selected method into the Clipboard. To paste the argument list into text, use Paste. Argument List retrieves the prototype argument list from the compiler results database.

Key shortcut: **Command-'**.

If you want to call a method and you don't know its arguments, this is a way to get them. Search through the active project until you find the method you are looking for, select text naming the method's or the source record icon, and then issue Argument List. You can then paste the prototype argument list for that method into text, where you can modify it to your needs.

For example, if you want to call the `remove` function, you can highlight `remove` in text and then choose Argument List to copy its argument list into the Clipboard. The following prototype argument list is copied to the Clipboard for `remove`:

```
(sequence, value, test: test, count: count)
```

You can then paste these arguments into place in a source record, in a text file, or in the Listener.

Related commands: Copy Special, Argument List (Insert Special).

See "Editing in Apple Dylan" on page 66.

Argument List (Insert Special) Edit

Inserts the prototype argument list for the function named by the selected text right after that function. Argument List retrieves the prototype argument list from the compiler results database. Argument List (Insert Special) is only available in the active project when you select text that names a compiled function or its source record icon.

Argument List (Insert Special) is accessible by pressing the Option key when clicking on Copy Special from the Edit menu.

Key shortcut: **Command-Option-'**

For example, if you select the function `draw`, Argument List (Insert Special) inserts the following prototype argument list directly after it:

```
(view, r)
```

Argument List (Insert Special) is handy when you are creating a new source record and you already know the name of the method you want to call. Simply type its name, select it, and then initiate Argument List (Insert Special).

This command does not replace the selection as the other Insert Special commands do.

Related commands: Insert Special, Argument List (Copy Special).

See "Editing in Apple Dylan" on page 66.

Aspect — Browse

Displays a list of aspects for an object (or objects). You can change the aspect for the basis of a pane, thus changing what is displayed in the pane, or simply display the aspect inline for a selected object.

▲ **WARNING**

Aspect only works on the active project.

Every object in Apple Dylan has applicable aspects. Aspects express the relationships between objects in Apple Dylan, such as what calls what, class relationships, methods, or families of functions. A source record's aspects, for example, include its source code and also any warnings that it has generated. An inheritance tree is an aspect of a class. Many objects, including all the variables in the active project, have the aspects References To and References From, and so on.

Key shortcut: **Command-option-click** on an object or in a pane without selecting anything.

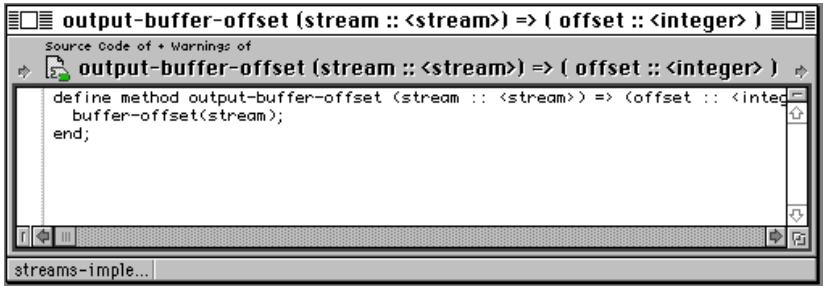
You can configure the panes in browsers to display various aspects of objects, either by changing existing browsers or creating new ones. However, Apple Dylan includes a number of built-in browsers that display many different aspects. They are on the Browse menu. If you don't see a built-in browser for the aspect you are interested in, use this command to change the aspect to what you want.

When no object in a pane is selected and this command is used, the aspects for the basis of the pane are displayed. Each browser pane displays the aspect that is the default aspect of the pane's basis. If you change the basis of the pane, you might see another aspect displayed since the new basis might have a different

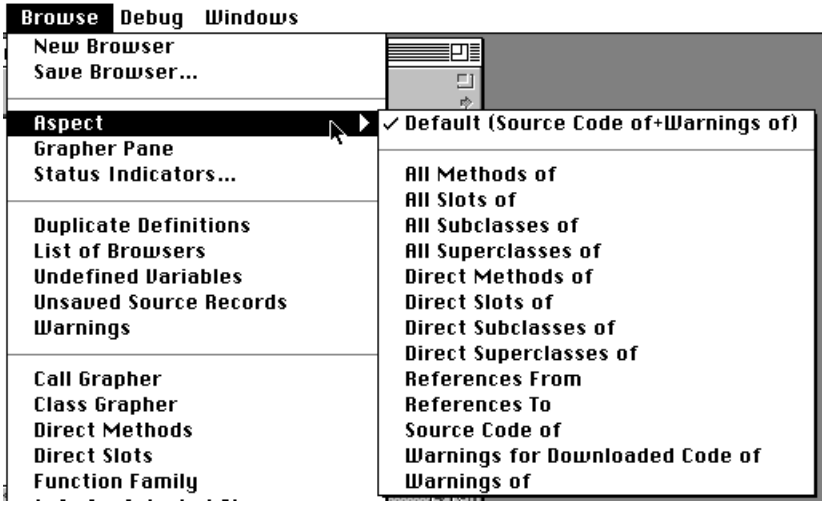
default aspect, but you can also change the aspect by overriding the default aspect. You do that by choosing another aspect from the list that is displayed when you use this command.

The name of the default aspect assigned to the object appears on the first line on the list in parentheses. You cannot change the list of aspects available for an object or its default aspect. The default aspect also dictates the aspect that is displayed in a new browser opened after an object is selected.

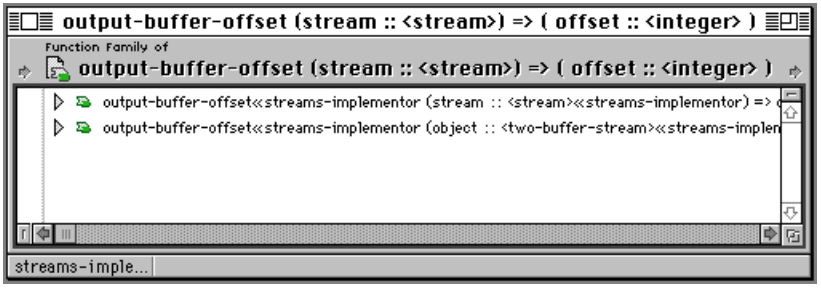
In the following example, a source record, output-buffer-offset, has been double-clicked to open a new browser with it as the basis. The source record's default aspect, Source Code of +Warnings of, is the aspect on view in the pane, as is noted in the pane's header (there are no warnings at the moment, so none appear, only source code).



The following figure shows the list of aspects available for source records. This list can be generated by selecting a source record, such as output-buffer-offset, and then using the Aspect command.



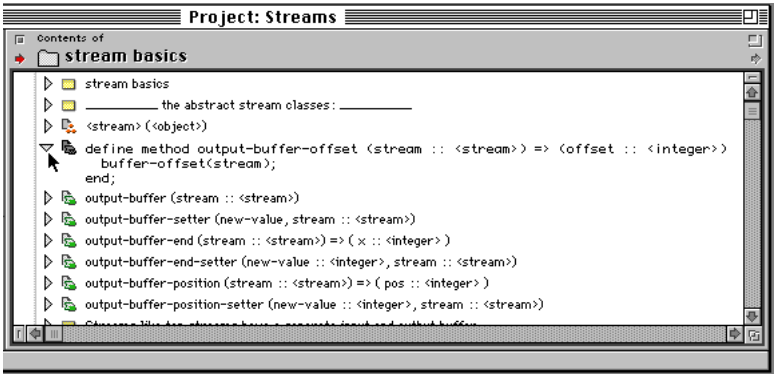
From the list of aspects, you can choose another aspect, such as Function Family. The pane now displays this aspect, as is shown in the following figure. The basis of the pane remains the same, but the name of the aspect in the header is changed to Function Family and the function family is shown within the pane.



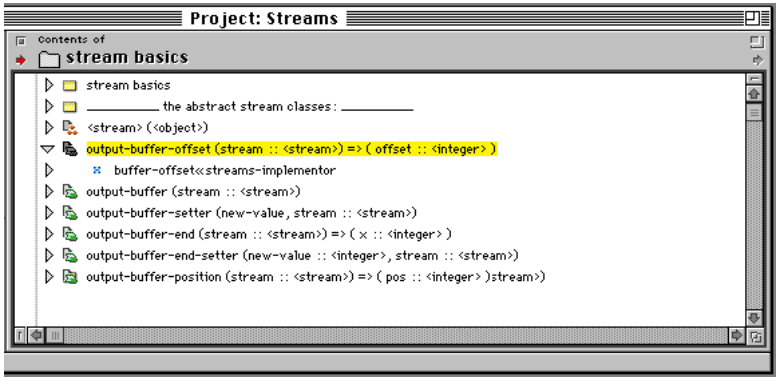
If you select an object in a pane and use this command, the aspects for that object are listed rather than those for the basis of the pane. The list of aspects displayed is all those aspects that apply to the selected object and only those aspects. If you select multiple objects, you see the union of aspects for all the objects. In this case, if an aspect does not apply to one of the multiple objects selected, you receive a message saying that the aspect is unavailable for that object.

When you expand an object by clicking its disclosure triangle, one aspect of the object is displayed inline below the name of the object. For instance, source records expand to reveal their source code right below the name of the source record. No new pane or browser opens. The aspect revealed is the default aspect, unless you select another using this command.

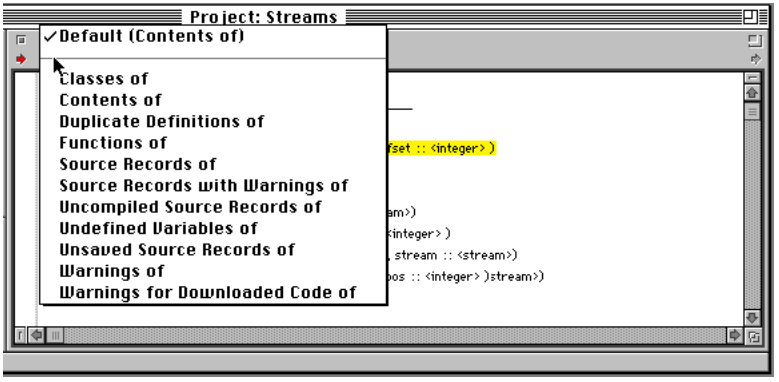
For example, in the following figure the source record output-buffer-offset is expanded inline to show its default aspect, Source Code+Warnings.



In the following figure, the Aspect command has been used to show the References From aspect for output-buffer-offset inline. Notice that the aspect listed on the pane's header is still Contents Of because the basis of the pane, the source folder stream basics, has not changed.



You can also see the list of aspects for a pane's basis by holding down command-option while pressing on the name of the pane's basis in the pane's header. The following figure shows the pop-up list for the source folder stream basics. It was generated by clicking within the right-hand pane without selecting anything, pressing command-option, and then holding down the mouse button with the cursor on the name "stream basics" in the pane's header. This pop-up list is the same as would be generated by using the Aspect command on the Browse menu.



If you change the aspect of an object and wish to reset it to the default, choose Default from the list. You can reset to the default aspects of several objects at once by selecting all of them and then choosing Default from the list. If you use Select All and then choose Default, the original default aspects of all browser panes are restored.

The following table shows all possible aspects and the objects to which they apply. You can create a custom browser pane displaying any aspect of any object or objects that have that aspect.

Aspect	Objects
All Methods of	class
All Slots of	class
All Subclasses of	class
All Superclasses of	class
Classes of	module, project, source folder
Contents	module, project, source folder
Direct Methods of	class
Direct Subclass of	class
Direct Superclasses of	class
Direct Slots of	class
Duplicate Definitions of	module, project, subproject, source folder
Function Family	generic function, method
Functions of	container
Modules of	project, subproject
References From	class, constant, function, interface, macro, method, slot, top level form, variable
References To	class, constant, function, interface, macro, method, slot, top level form, variable, warning
Resource Files of	project, subproject
Source Code of	class, constant, function, interface, library, macro, method, module, slot, top level form, variable
Source Folders	module, project, subproject
Source Records	module, project, subproject, source folder
Source Records with Warnings	project, subproject, module, source folder

continued

Aspect	Objects
Subprojects	project
Text of	error
Uncompiled Modules of	project, subproject
Uncompiled Source Folders	module, project, subproject
Uncompiled Source Records	project, subproject, module, source folder
Undefined Variables	module, project, subproject, source folder
Unsaved Modules of	project, subproject
Unsaved Source Folders	module, project, subproject
Unsaved Source Records	module, project, subproject, source folder
Variable Definitions of	module, project, subproject, source folder
Warnings for Downloaded code of	class, constant, function, interface, macro, method, module, project, subproject, slot, source folder, top level form, variable
Warnings of	class, constant, function, interface, macro, method, module, project, subproject, slot, source folder, top level form, variable
Warning Source Record of	warning

Related commands: all the aspects listed in the table of aspects.

See “Showing different aspects of objects” on page 42 and“Changing aspects” on page 46.

Break — Debug

Stops an application from running without untethering from the development environment. No state is lost; everything is frozen. The action is the same as if the code had executed `break ()`.

Key shortcut: **Command-,**

Break starts a break loop one level greater than the one in which the current code was executing. You can resume running the application with Continue or Abort.

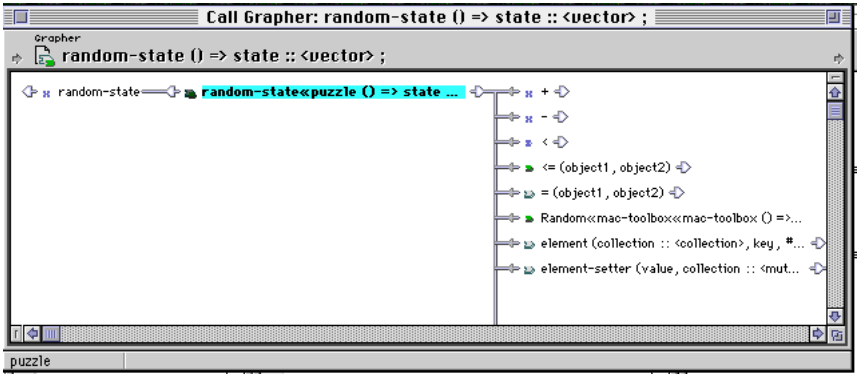
Note
Break levels are recursive. This is because inspecting and altering values is executing code in the Listener. It is possible for that code to have error or break statements, so if the new code stops, you enter another break level, which you can use to debug the code that inspected or altered values that you ran in the previous break level.

Call Grapher — Browse

Works on methods, opening a browser presenting a graphical representation of a function's calling relationships. Click on the arrows on the graph to expand and collapse it.

Related commands: Class Grapher, Grapher Pane.

The following figure shows the Call Grapher browser for the `<random-state>` method of the sample project puzzle.



In the figure, the method `<random-state>` is the basis of the Call Grapher browser. Therefore, while you can go up and down the graph from `<random-state>`, full information is shown only for `<random-state>`. Control-click on another object to make it the basis of the browser so you can see full information on it.

See the task “Using the browser Info for Selected Class” on page 53 for more information on using grapher panes.

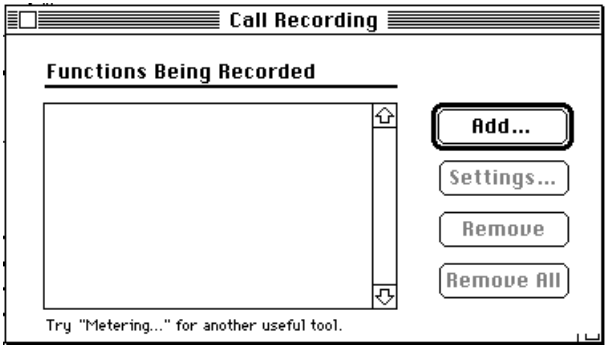
Call Recording Debug

Records the behavior of specific functions in the runtime. The results of Call Recording are printed to the Listener. You must be tethered to the development environment for Call Recording to work.

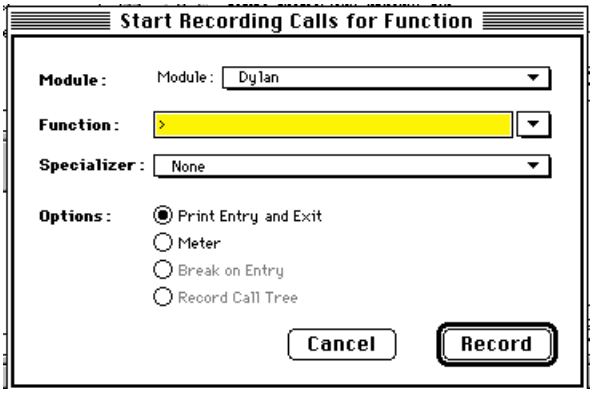
For more basic information about all the generic functions and classes in a project, use Meter Expression.

Related command: Meter Expression.

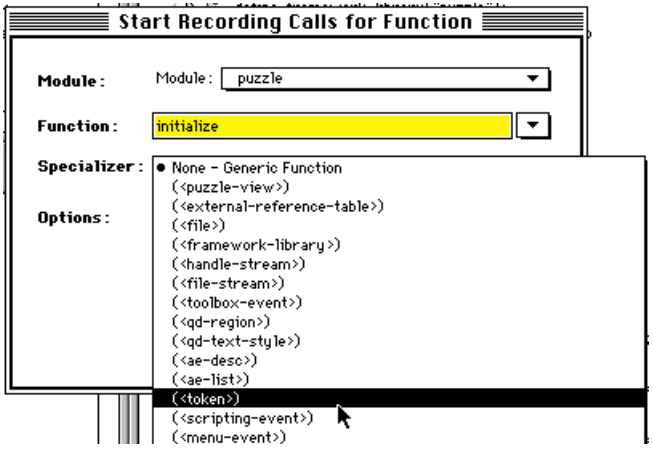
The following figure shows the Call Recording dialog box. This is where the recorded functions are listed after they are chosen. To select a function to add to the list, choose Add.



The Start Recording Calls for Function dialog box opens.

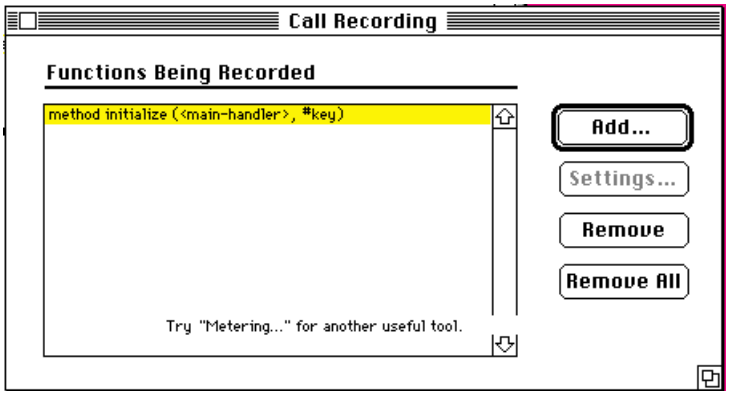


You then select the module the function is in, the name of the function, and the specializer you want to record. The following figure shows some of the specializers you could choose for the function “initialize” in the puzzle module.



If you want to record a function and print the results to the Listener, choose Print Entry and Exit from the Options field. If you want to meter a function, choose Meter from the Options field.

When you have made your choices, click Record. The results are printed to the Listener and the recorded function is listed in the Call Recording window.



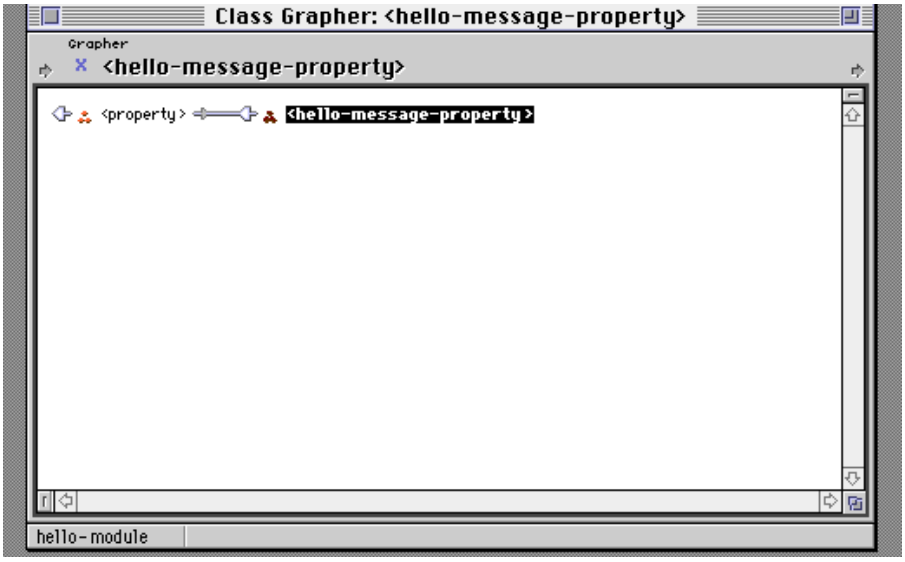
The Remove button deletes a selected function from the list. The Remove All button deletes all the functions from this list.

Class Grapher — Browse

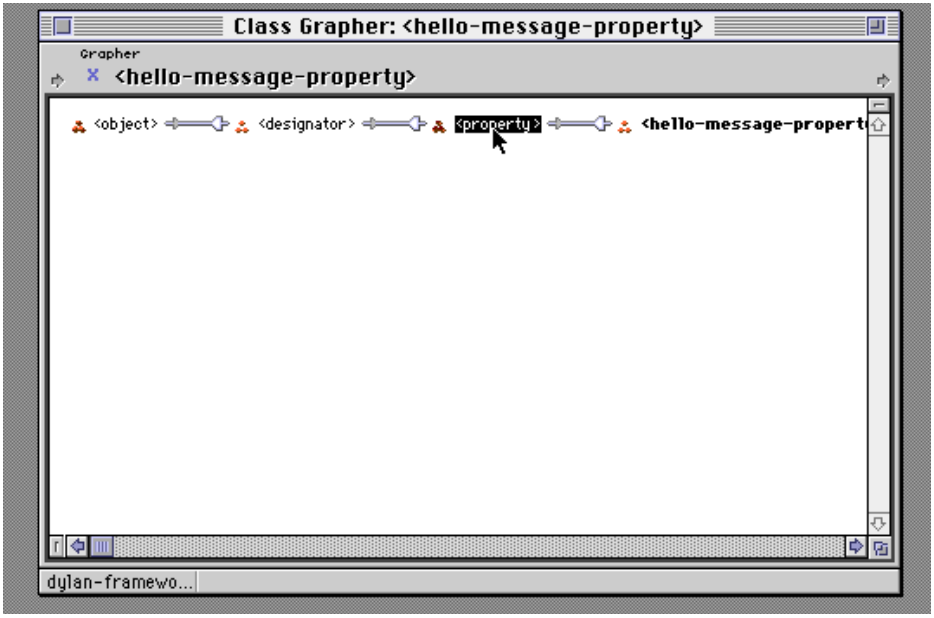
Operates on classes, opening a browser that graphically depicts the superclasses and subclasses for a class. Click on the arrows on the graph to expand and collapse it.

In the following figure, the class `<hello-message-property>` is the basis of the class grapher browser. Therefore, while you can go up and down the graph from `<hello-message-property>`, full information, specifically multiple inheritance and subclass relations, are shown only for the `<hello-message-property>`. Control-click on another object to make it the basis so you can see its multiple inheritance and subclass relations.

In this example, two classes are displayed, <hello-message-property> and its superclass <property>. You can view any superclasses of <property> by clicking on the arrow to its left.



Continue clicking on the left-most arrow until you reach the base class. If you click on the arrows on the right end of the bars that separate each class, you can collapse the graph, hiding the superclasses.



Related commands: Call Grapher, Grapher Pane, Info for Selected Class.

See the task "Using the browser Info for Selected Class" on page 53 for more information on using grapher panes.

Class Template (Copy Special) Edit

Creates a prototype class template named `<foo>`, which is a subclass of the selected class, and copies the template into the Clipboard. To paste the class template into text, use Paste. You can select either text that names a class or the source record icon for the class from the active project.

For example, if you select the class `<number>`, Class Template (Copy Special) copies the following template into the Clipboard:

```
define class <foo> (<number>)
end class;
```

You can then paste it into text.

You can select several classes at once and then use Class Template (Copy Special) to create a class that is a subclass of all the selected classes. For example, if you select the classes `<cell>` and `<cell-view>` and then choose Class Template (Copy Special), you copy the following template to the Clipboard:

```
define class <foo> (<cell>, <cell-view)
end class;
```

Note that this command appears in the menu only if you select a class. If you select a generic function or method, Method Template (Copy Special) appears instead.

Key shortcut: **Command-T**

Related Commands: Copy Special, Class Template (Insert Special), Method Template (Copy Special).

See "Editing in Apple Dylan" on page 66.

Class Template (Insert Special) Edit

Replaces the selected class with the prototype template of a subclass for the selected class. Class Template (Insert Special) appears in the menu only if you select editable text in the active project that names a class.

For example, if you select the class `<bar>`, Class Template (Insert Special) replaces it with the following text:

```
define class <foo> (<bar>)

end class;
```

If you want to create a subclass based on a class, simply type its name, select it, and then initiate Class Template (Insert Special).

This command is made available by pressing Option when clicking on Copy Special on the Edit menu.

Key shortcut: **Command-Option-T**

Note
Class Template (Insert Special) creates a subclass based on a single class. Class Template (Copy Special) creates a subclass based on one or more classes.

If you select a generic function or method, Method Template (Insert Special) appears in the menu instead of Class Template.

Class Template retrieves the prototype template for the selected object from the compiler results database and inserts it into a source record.

Related Commands: Insert Special, Class Template (Copy Special), Method Template (Copy Special).

See "Editing in Apple Dylan" on page 66.

Classes of (Aspect) — Browse

Operates on a container in the active project, that is, any object that contains other Apple Dylan objects, such as projects, subprojects, modules, and source folders, displaying its classes.

Classes of (Aspect) is commonly used to customize a browser pane displaying that aspect of a container. You can also select a single container in a pane and display all its classes inline.

See Aspect and "Customizing browsers" on page 58 for more information.

Clear — Edit

Deletes the selected text or object. The deleted text is not copied to the Clipboard. You can use Undo Clear to reinstate the cleared text. Undo, Clear, Redo Clear, and Undo Clear replace one another on the File menu, depending on which is appropriate.

See "Editing in Apple Dylan" on page 66.

Pressing Delete or inserting text when there is a highlighted region present is the same as doing a Clear.

▲ **WARNING**

Clear supports Undo and Redo only for text. If you have cleared an object, folder, or subproject, it cannot be undone or redone. If you are about to clear a subproject, you are warned. You cannot clear required subprojects.

Close — File

Closes a browser, window or project. Closing the project window closes the project and all of its subprojects, unless the subprojects are used in another project that is still open. You are prompted to save changes, if any. Closing the active project while unrelated projects are still open makes the next project the active project. You can close all open windows and browsers except the project browser by holding down the Option key when you choose Close.

If you have a window active and hold down the Option key when you choose the Close command, all open windows of the same type, such as browsers or inspector windows, are closed. However, the project browser is not closed with the Option-Close command.

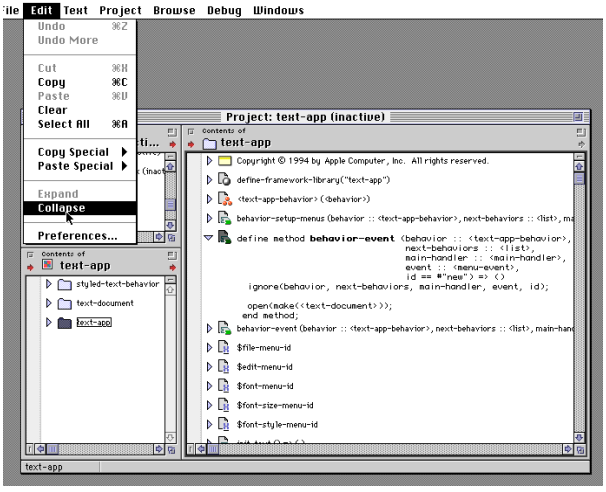
Key shortcut: **Command-W**

Collapse — Edit

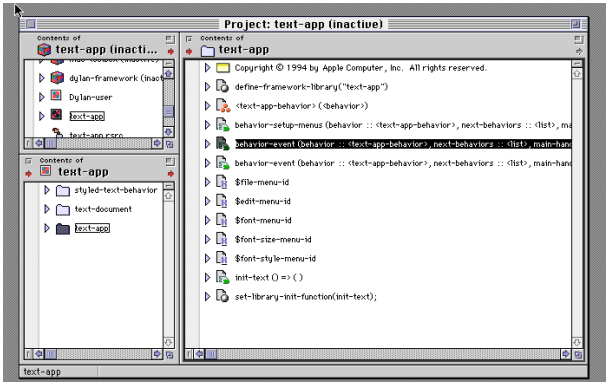
Collapses the selected object so you cannot see its contents. You can also collapse an object by clicking on its disclosure triangle, which is the triangle to the left of the object's icon. You can collapse several objects at once by selecting them and choosing Collapse.

Related command: Expand.

The following figure shows the object "behavior-event" selected and Collapse about to be chosen.



The following figure shows the object “behavior-event” collapsed.



Color — Text

Changes the color of the selected source text. This choice changes only the text in a source record and has no semantic impact but it does flag the record for recompilation. You can use color to help organize your sources.

Compact Project — Project

Reduces the disk space used by a project by eliminating unused information from the source database by compacting the files in the selected open project.

The actions initiated include quitting the Application Nub (for the active project), closing all project windows, and then compacting the files. You are prompted to save your project if you have not. When compaction is complete, the project is automatically reopened.

Compaction is a form of garbage collection for Apple Dylan source databases. These databases retain everything that has been written to them until compacted. This old information is of no value because it is not accessible, but it is still occupying disk space.

See "Apple Dylan User Model" on page 95.

Compile and Download Region — Project

Compiles an extended area within a single source record of the active project and downloads it to the Application Nub. This command replaces Compile Region on the menu when the Application Nub has been launched.

Key shortcut: **Command-E**

Compile and Download Region appears on the menu only if you select the text by highlighting it and if the text exceeds more than a single expression. In a pane, if you place the cursor in the text, the Compile and Download Selection command appears on the menu instead. In the Listener, if you place the cursor in the text, Compile Expression appears instead.

Related commands: Compile and Download Selection, Compile Expression, Compile Selection, Recompile, Update Project.

Compile and Download Selection — Project

Compiles the selected source record in a pane in the active project or highlighted code in the Listener and downloads the result to the Application Nub. This command replaces Compile Selection on the menu when the Application Nub has been launched.

Key shortcut: **Command-E**

To select a source record, select its icon rather than its name or text. You can select more than one source record. Source record objects are methods, classes, variables, constants, macros, generic functions, comments, and top level forms.

To compile code within a source record, you can highlight the code you want to compile, in which case Compile and Download Region replaces Compile and Download Selection on the menu.

Compile and Download Selection can also compile commented out code if you select the code within the range of the comment.

In general, Command-E compiles whatever is there to be compiled.

Related commands: Compile and Download Region, Compile Expression, Compile Region, Recompile, Update Project.

Compile Expression — Project

Compiles the selected expression and downloads (only when tethered) the result to the Application Nub from the Listener. Compile Expression appears only if you select the expression by placing your cursor in it; if you highlight one or more expressions, Compile Selection appears instead. An expression is any code that returns a value.

This command works only in the Listener with the insertion point within the expression and with nothing selected.

Key shortcut: **Command-E**

Related commands: Compile Region, Compile Selection, Recompile, Update Project.

See “Apple Dylan User Model” on page 95.

Compile Region — Project

Compiles an extended area within a single source record. This command replaces Compile and Download Region on the menu if the Application Nub has not been launched.

Key shortcut: **Command-E**

Compile Region appears on the menu only if you select the text by highlighting it and if the text exceeds more than a single expression. In a pane, if you place

the cursor in the text, the Compile Selection command appears on the menu instead. In the Listener, if you place the cursor in the text, Compile Expression appears instead.

Related commands: Compile and Download Region, Compile Expression, Compile Selection, Recompile, Update Project.

Compile Selection — Project

Compiles the selected source record in a pane or highlighted code in the Listener. This command replaces Compile and Download Selection on the menu if the Application Nub has not been launched.

Key shortcut: **Command-E**

To select a source record, select its icon rather than its name or text. You can select more than one source record. Source record objects are methods, classes, variables, constants, macros, generic functions, comments, and top level forms.

To compile code within a source record, you can highlight the code you want to compile, in which case Compile Region replaces Compile Selection on the menu.

Compile Selection can also compile commented out code if you select the code within the range of the comment.

You can use Compile Selection from the Apple Dylan Listener window by highlighting the code you want to compile.

In general, Command-E compiles whatever is there to be compiled.

Related commands: Compile and Download Selection, Compile Expression, Compile Region, Recompile, Update Project.

Contents of (Aspect) — Browse

Operates on a container in the active project, that is, any object that contains other Apple Dylan objects, such as projects, subprojects, modules, and source folders, displaying its contents.

Contents of is the default aspect of most panes in the browsers supplied as part of the Apple Dylan development environment.

Continue — Debug

Resumes running an application that has been stopped with a Break.

Key shortcut: **Command-/**

Related command: Break.

Copy — Edit

Copies the selected text (or text of the selected object) to the Clipboard. You can then use Paste to insert the contents of the Clipboard into a source record or the Listener.

Key shortcut: **Command-C**

Related commands: Cut, Paste, Copy Special.

See "Editing in Apple Dylan" on page 66.

Copy Special — Edit

Displays a menu containing two of three possible commands: Argument List and either Class Template or Method Template.

The Copy Special commands are enabled only if a project is active and you select a valid object. The object can be text that names an object (for example, a method name) or a source record icon. The commands copy the prototype argument list or template to the Clipboard. You then use Paste (from the Edit menu) to paste the contents of the Clipboard into a source record or the Listener.

The Copy Special commands retrieve the prototype argument list or template for the selected object from the compiler results database and write it to the Clipboard.

Key shortcuts: **Command-'** for just the argument list or **Command-T** for the templates.

The closely related Insert Special commands are available by pressing Option while clicking on the Copy Special menu. Where the Copy Special commands write to the Clipboard, the Insert Special commands write into the current buffer, including the Listener.

See: Argument List (Copy Special), Class Template (Copy Special), and Method Template (Copy Special).

Related commands: Insert Special.

See "Editing in Apple Dylan" on page 66.

Note
The Copy Special and Insert Special commands are useful for getting information as well as for writing code.

Copy Title Text — Edit

Copies the title text of selected objects other than source records, such as modules and subprojects. When you select valid title text, Copy Title Text appears; otherwise, Copy appears.

Related commands: Copy, Paste.

See "Editing in Apple Dylan" on page 66.

Create Application — Project

Builds a stand-alone application from your active project. Give your application the name you want to appear in the Finder and choose its location.

You must have issued Set Project Type and quit the Application Nub (See Quit Application Nub) before issuing Create Application. It is also suggested that you issue Recompile before issuing Create Application.

When you choose Save on the Create Application dialog box, the application is updated first and then built. If "Use separately loaded libraries" was checked in the Set Project Type dialog, a copy of the link library is placed in the system folder.

If you save an application to the desktop or an open folder, it appears as a document rather than an application. To make it appear as an application, you can either move it to a closed folder or close the folder it's in and then reopen it.

If you are building a library instead on an application, Create Library appears on the Project menu instead of Create Application. You designate which you are building using Set Project Type on the Project menu.

Related commands: Update Project, Create Library, Set Project Type.

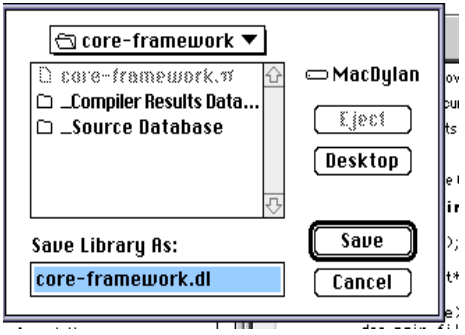
Create Library — Project

Builds a library from your active project. You must Set Project Type and untether from the development environment (See Quit Application Nub) before issuing Create Library. You should give your library the same name you entered in its Library ID field when you used Set Project Type.

If you are building an application instead of a library, Create Application appears on the Project menu instead of Create Library. You designate which you are building using Set Project Type on the Project menu.

Create Library creates both a _Library Model document, used by the development environment, and a .dl document, used by the application itself.

When you choose Save, the library is updated first and then built.



Related commands: Update Project, Create Application.

Cut — Edit

Deletes the selected text and saves it to the Clipboard.

Key shortcut: **Command-X**

Related commands: Copy, Paste.

See “Editing in Apple Dylan” on page 66.

Direct Methods of — Browse

Operates on a selected class. Displays a browser with the direct methods for the class.

Related commands: Direct Methods of (Aspect), All Methods of (Aspect).

Direct Methods of (Aspect) — Browse

Operates on a selected class in the active project, displaying its direct methods only.

Direct Methods of (Aspect) is commonly used to customize a browser pane displaying that aspect of a class. You can also select a single class in a pane and display its direct methods inline.

See Aspect and "Customizing browsers" on page 58 for more information.

The Direct Methods of command opens a single-pane browser displaying direct methods.

Related commands: All Methods of (Aspect).

Direct Slots of — Browse

Operates on a selected class. Displays a browser with the direct slots for the class.

Related commands: Direct Slots of (Aspect), All Slots of (Aspect).

Direct Slots of (Aspect) — Browse

Operates on a selected class in the active project, displaying its direct slots only.

Direct Slots of (Aspect) is commonly used to customize a browser pane displaying that aspect of a class. You can also select a single class in a pane and display its direct slots inline. See Aspect and "Customizing browsers" on page 58 for more information.

The Direct Slots of command opens a single-pane browser displaying direct methods.

Related command: All Slots of (Aspect).

Direct Subclass of (Aspect) Browse

Operates on a class in the active project, displaying its direct subclasses only.

Direct Subclasses of (Aspect) is commonly used to customize a browser pane displaying that aspect of a class. You can also select a single class in a pane and display its direct subclasses inline.

See Aspect and "Customizing browsers" on page 58 for more information.

Related commands: Class Grapher, All Subclasses of (Aspect), All Superclasses of (Aspect), Direct Superclasses of (Aspect).

Direct Superclasses of (Aspect) Browse

Operates on a class in the active project, displaying its direct superclasses only.

Direct Superclasses of (Aspect) is commonly used to customize a browser pane displaying that aspect of a class. You can also select a single class in a pane and display its direct superclasses inline.

See Aspect and "Customizing browsers" on page 58 for more information.

Related commands: Class Grapher, All Superclasses of (Aspect), All Subclasses of (Aspect), Direct Subclass of (Aspect).

Duplicate Definitions Browse

Operates on a container in the active project, that is, any object that contains other Apple Dylan objects, such as projects, subprojects, modules, and source folders.

This command opens a browser that lists the duplicate definitions in the container.

For most objects, having the same name is sufficient to define duplicates, but in the case of methods, duplicate definitions have not only the same methods, but the same type signatures.

Related command: Duplicate Definitions of (Aspect).

Duplicate Definitions of (Aspect) — Browse

Operates on a container in the active project, that is, any object that contains other Apple Dylan objects, such as projects, subprojects, modules, and source folders, displaying its duplicate definitions.

For most objects, having the same name is sufficient to define duplicates, but in the case of methods, duplicate definitions have not only the same methods, but the same type signatures.

The Duplicate Definitions command opens a single-pane browser displaying duplicate definitions.

Duplicate Definitions of (Aspect) is commonly used to customize a browser pane displaying that aspect of a class. You can also select a single class in a pane and display its duplicate definitions inline.

See Aspect and "Customizing browsers" on page 58 for more information.

Exclude Source Records — Project

Marks a selected source record as excluded so it won't be compiled or included in an update. Excluded source records are not downloaded when the project is loaded into the Application Nub. Excluded code is not included in your application or library when you build it.

Related command: Include Source Records.

Examples of source records that might be marked as excluded: test code, unfinished code, unused alternate code, or example code, that is, anything not ready or suitable for compilation and use.

Excluding a source record is equivalent to commenting it out, except that:

- Its icon and name show the kind of source record it is
- It can be browsed
- It can be switched from Excluded to Included without editing.

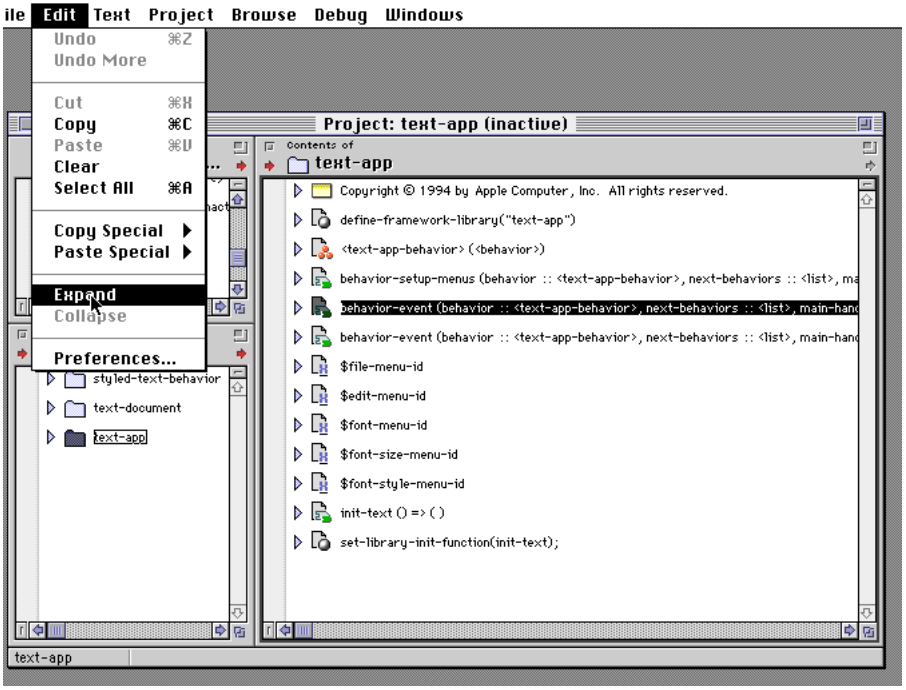
Excluded source records can be compiled by selecting them and pressing Command-E.

Expand Edit

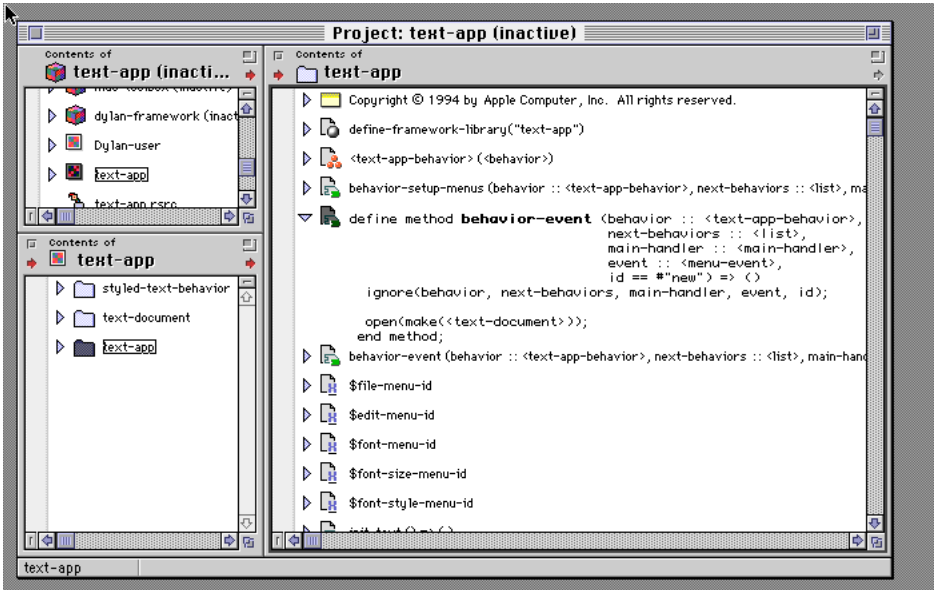
Expands the selected object so you can see its contents. To expand an object you click on its icon and then choose Expand. The object's contents then appear below its name. You can also expand an object by clicking on its disclosure triangle, which is the triangle to the left of the object's icon. You can expand several objects by selecting their icons and then choosing the Expand command.

Related command: Collapse.

The following figure shows the object "behavior-event" about to be expanded.



The following figure shows it expanded.



Expand Macro — Debug

Prints the full macro expansion, displaying exactly the text sent to the compiler by the parser. This command is used in debugging macros.

Operates on statement macros and definition macros. Does not work on function macros. This command does not work on the icon; you must select the full text of the macro in a browser or in the Listener. Expands the selected macro to the Listener. You can also select the macro text in the Listener.

Key shortcut: **Command-M**

You can limit the expansion to one level of macro expansion by pressing the option key before clicking on Expand Macro.

Key shortcut: **Command-option-M**

Related command: Pattern Match Macro.

Export File

Exports the selected source folder from the development environment, along with everything in the source folder. You cannot export an individual source record. Export creates a Dylan text file with the suffix ".dylan".

Not every object can be exported and imported without changes.

- A single source record with more than one top-level form or comment is broken into an individual source record for each top level form or comment when exported.
- In addition, if any code within a source folder has the incorrect number of begins or ends, or is any other way seriously malformed, the source folder can be exported, but the resulting Dylan text file cannot be imported.

The exported object is stored in a Dylan text file, which has the suffix ".dylan". The Dylan text file has a header. Dylan text files can be edited outside of Apple Dylan in other word processors or editors and mailed electronically.

Related command: Import.

Importing or exporting has no effect on objects marked for exclusion or inclusion. Markings remain as they were.

Find/Replace Text

Finds and optionally replaces the text you specify in the Find&Replace dialog box. This command searches through source code only, not the names of containers, modules, etc. Find/Replace opens the object containing the specified text. You can choose whether to search the current pane or the entire project. If you choose to search the entire project, opens a new browser. You may choose simply to find the text, find and replace one instance of the text, or find and replace all instances.

Key shortcut: **Command-F**

See "Editing in Apple Dylan" on page 66.

Find Again — Text

Searches for whatever you last searched for using Find/Replace. Use Replace & Find to change the target text and search for the next instance.

Key shortcut: **Command-G**

See "Editing in Apple Dylan" on page 66.

Find Selection — Text

Searches for a selected string. Find Selection finds only and has no effect on the settings of Find/Replace, Find Again, or Replace & Find, nor does Find Selection support replacement.

Key shortcut: **Command-H**

See "Editing in Apple Dylan" on page 66.

Font — Text

Changes the font of the selected source text. This choice changes only the text in a source record and has no semantic impact, but it does flag the source record for recompilation. You can use fonts to help organize your sources.

See "Formatting commands" on page 70.

You can change the default font for the Listener and browsers, as well as the font for editing code, using Preferences.

Function Family — Browse

Operates on a selected method or generic function.

Opens a browser that lists the family of methods and the generic function the selected method belongs to. You can have more than one Functions browser open at once. If no object is selected, the functions for the entire project are listed.

Related command: Function Family (Aspect).

Function Family (Aspect) Browse

Operates on a selected method or generic function in the active project, displaying all method definitions as well as the general function definition (if one exists) for the method or function.

Here is how it works in each case:

- If a method has been selected, finds generic function it is associated with along with all other methods of that generic function
- If a generic function has been selected, finds all methods associated with that generic function.

Function Family (Aspect) is commonly used to customize a browser pane displaying that aspect of a class. You can also select a single class in a pane and display its function family inline.

See Aspect and "Customizing browsers" on page 58 for more information.

The Function Family command opens a single-pane browser displaying direct methods.

Functions of (Aspect) Browse

Operates on a module in the active project, displaying all its functions. If no object is selected, the functions for the entire project are listed.

Functions of (Aspect) is commonly used to customize a browser pane displaying that aspect of a class. You can also select a single class in a pane and display its functions inline.

See Aspect and "Customizing browsers" on page 58 for more information.

Related commands: Function Family, Function Family (Aspect).

Grapher Pane Browse

Changes the active pane to a graph showing the relationship of the objects in pane. If the basis of the pane is a class, the class's superclasses and subclasses are graphed. If the basis of the pane is a function, the function's callers and callees are graphed. Click on this command on the Browse menu to deselect it and return the pane to its original type.

Full information, such as multiple inheritance, is shown only for the object that is the basis of the pane. Click on the arrows on the graph to expand and collapse it.

Only a single relationship is shown for other objects in the pane. Control-click on another object to make it the basis so you can see full information on it.

Related commands: References To, References From, Call Grapher, Class Grapher, Info for Selected Class.

See the task "Using the browser Info for Selected Class" on page 53 for more information on using grapher panes.

Hide Interface Builder — Apple

Hides the Apple Dylan interface builder to return you to the development environment. It's a good idea to save your builder project before hiding it. Note that this command does not quit the interface builder, it just hides it.

For more information on the interface builder, see the book *Creating a User Interface in Apple Dylan*.

Related commands: Show Interface Builder, Load UI Builder.

Import — File

Imports Dylan text files. Dylan text files are created using a word processor or editor or by exporting a source folder from Apple Dylan and have the suffix ".dylan".

If any exported code within a source folder has the incorrect number of begins or ends, or has partial source records, the source folder cannot be imported.

Related commands: Export, New Text Window.

See "Importing and exporting Dylan text files" on page 70.

Importing or exporting has no effect on objects marked for exclusion or inclusion. Markings remain as they were.

Include Source Records — Project

Changes excluded source records into included source records. Includes a specified source record with each compilation and update. Included code is

automatically recompiled and updated when those commands are issued. Source records are automatically marked as included when they are created.

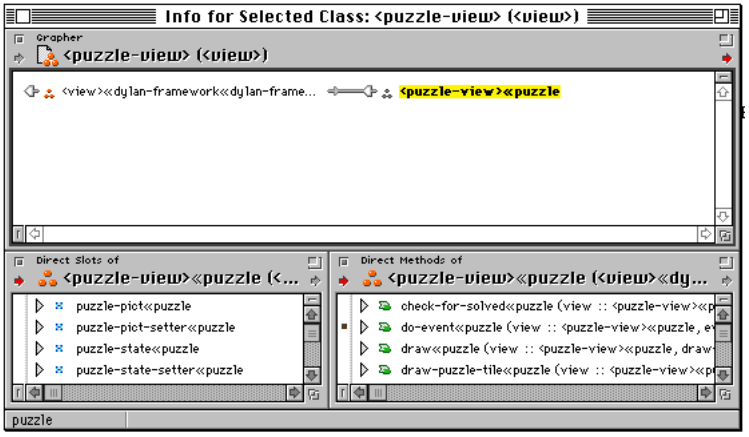
Related command: Exclude Source Records.

Info for Selected Class — Browse

Operates on a class, opening a three-pane browser that shows the selected class's hierarchy in the grapher pane, its directs slots in the lower left pane, and its direct methods in the lower right pane. Click on the arrows on the graph to expand and collapse it.

Full information, such as multiple inheritance, is available only for the basis of the browser. Only a single relationship is shown for other classes in the pane. Control-click on another class to make it the basis so you can see full information on it.

The following figure shows the Info for Selected Class browser with <puzzle-view> as it basis. The class <puzzle-view> has been selected, so its direct slots and direct methods are listed in the two lower panes.



Related commands: Grapher Pane, Call Grapher, Class Grapher.

See the task "Using the browser Info for Selected Class" on page 53 for more information.

Insert Special — Edit

Displays a menu containing two of three possible Insert Special commands: Insert Argument List and either Class Template or Method Template. The Insert Special commands work only on selected text, not on selected objects.

The Insert Special commands are available by pressing Option when clicking on Copy Special on the Edit menu.

Key shortcuts: **Command-Option-'** for just the argument list or **Command-Option-T** for the templates.

The Insert Special commands are enabled only if a project is active and you have selected text that names a method, generic function, or class. The type of selected text controls which Insert Special commands are available.

The Insert Special commands retrieve the prototype argument list or template for the selected object from the compiler results database and insert it into a source record or other buffer.

The closely related Copy Special commands write to the Clipboard, where the Insert Special commands write into the current buffer, including the Listener.

See Argument List (Insert Special), Class Template (Insert Special) and Method Template (Insert Special).

Related commands: Copy Special.

Note
The Insert Special and Copy Special commands are useful for getting information as well as for writing code.

Inspect Heaps — Debug

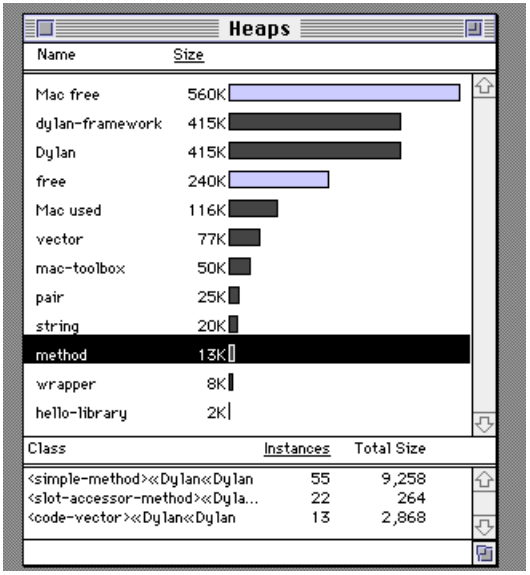
Opens an inspector window for inspecting the heaps when you want to monitor how your application is using memory. The heaps shown in this window are a snapshot of all the heaps in the runtime. These heaps are either downloaded with a project or created by it.

There are several types of heaps in Apple Dylan, each for a different type of object. The heaps are calculated for you. When you select a heap in the top pane, the contents are displayed in the bottom pane. The top pane is a snapshot of the heaps taken when Inspect Heaps was issued. The bottom pane is a

snapshot of the heap at the time you select the heap from the top pane. You can issue Inspect Heaps repeatedly to take snapshots at different times.

Note
Macintosh heaps are not shown in this window. Macintosh heaps, such as for windows and menus, are created by the toolbox and are not monitored by the development environment.

The following figure shows the heap inspection window with "method" selected in the top pane.



The gray bars indicate that memory has been allocated but is unused, while the black bars indicate the memory used. The Mac free and free heaps are always completely empty and the Mac used heap is always completely full. Memory moves from the Mac free heap into the Mac used heap, neither of which contains any Apple Dylan objects nor is under Apple Dylan control. The free heap is for use by Apple Dylan objects, but has not been allocated yet.

In the example above, two different libraries in the project show up separately. Library heaps cannot be grown or garbage collected. Libraries are displayed as

subprojects in the project window. Every project has at least one library heap for the Dylan subproject.

The other heaps contain Apple Dylan objects that have been dynamically allocated by the application and can be garbage collected. Most instances of user-defined classes go into the vector heap.

Use this window to find an object you couldn't find programmatically. If you double-click on a heap, a window opens listing of all the classes in that heap. If you then double-click on a class, an inspector window on that class opens. You can also double-click on a class in the bottom part of the Heaps window to open an inspector window on a class. See the description of Inspect Stack for more information on inspector windows.

You can sort each part of the Heaps window by clicking on a column's name. For instance, in the figure above, the bottom part is sorted by Instances, which is indicated by the underlining. The number of instances is always one greater than the actual number because the prototype instance for the class is included in this count. The Total Size column is in bytes.

Inspect Listener Result — Debug

Opens an inspector window on the first return value from the last form compiled and downloaded to the Application Nub, whether from the Listener or from Compile Selection. The command is available only when tethered.

Key shortcut: **Command-I**

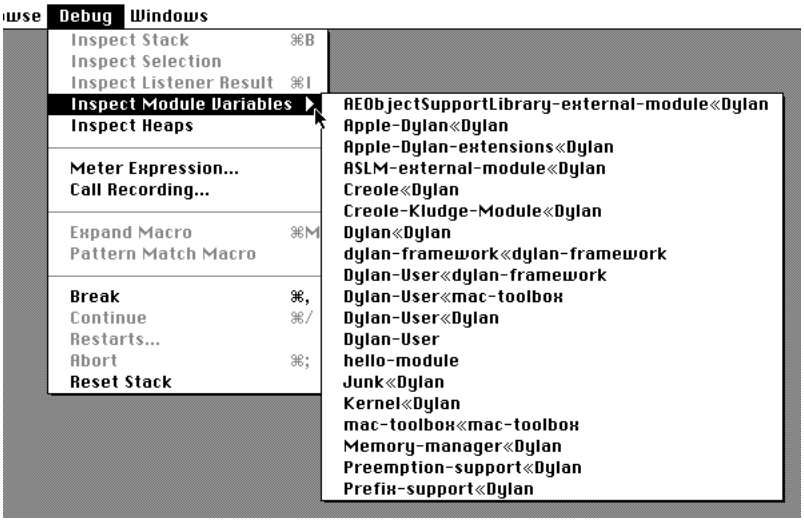
See the description of Inspect Stack for more information on inspector windows.

See "Inspecting Listener results" on page 151 for an example using this command.

Inspect Module Variables — Debug

Lists all the modules in the Application Nub. Before looking for a module on this list, you should use the Update Project command. This ensures that all the modules have been downloaded to the Application Nub. You choose a module from the list to open an inspector window for it.

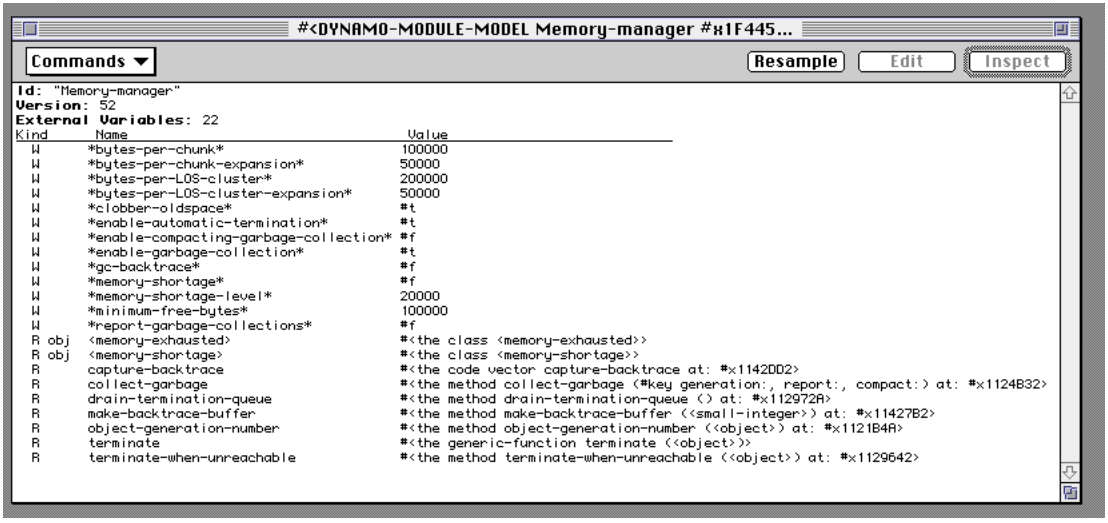
The following figure shows an example of the modules in a project's Application Nub.



The module inspector window displays the names and values of the variables that are defined in a module, as well as their read/write status. A menu item in the Commands popup menu toggles between "Show All Variables" and "Show Exported Variables."

Note

The module inspector window shows only variables that are actually created in that module. It does not show variables imported from other modules even if they are re-exported from the module being inspected. This is why, for instance, the inspector doesn't show any variables in the "Dylan" module.



In addition, the inspector window displays the kind of variable it is. Typically, variables created by “define-class” and some variables with type declarations are “obj” kind variables, which are slightly smaller than the other kinds.

Kinds of variables:

- blank—a variable that can hold an object or code.
- “obj”—a variable that can hold only objects.
- “jump”—a variable that can hold only code.
- “byte”—a variable that can hold only unboxed bytes.
- “shor”—a variable that can hold only unboxed shorts.
- “long”—a variable that can hold only unboxed longs.
- “doub”—a variable that can hold only unboxed doubles.
- “code”—a variable that can hold only unboxed code.

Double-clicking on a variable in the inspector window opens another inspector window on the value of the variable.

See the description of Inspect Stack for more information on inspector windows.

Inspect Selection — Debug

Opens an inspector window for the selected runtime object.

See the description of Inspect Stack for more information on inspector windows.

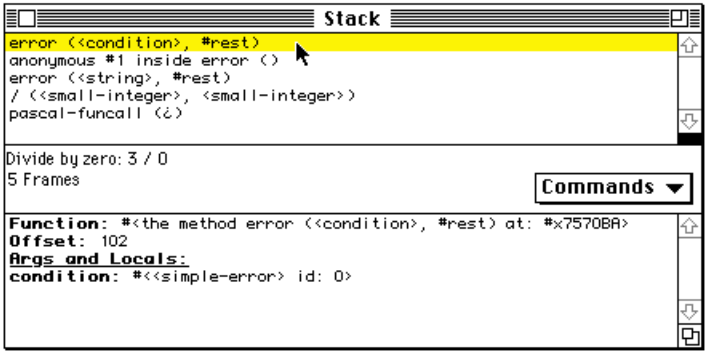
Inspect Stack — Project

Opens the Stack window, which allows you to inspect the current state of the stack after an error or break is signalled. There is roughly one stack frame per unreturned function call listed in the Stack window.

Key shortcut: **Command-B**

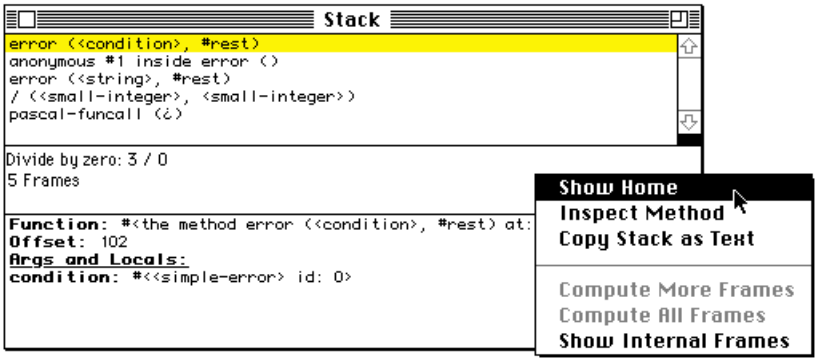
The Stack window is divided into three sections. The top pane lists the frames on the stack. The central section contains information about the error and the pull-down commands for the Stack window. As you select each frame in turn in the top pane, the object's active parameters and local variables appear in the bottom pane.

The following figure shows a typical Stack window with the first line in the top pane selected. Its contents appear in the bottom pane.



The following figure shows the commands for the top pane of the Stack window. The commands for the Stack window are Show Home, Inspect Method, Copy Stack as Text, Compute More Frames, Compute All Frames, and Hide/Show Internal Frames. These commands apply only to the top pane. The

bottom pane displays the active parameters and local variables for the frame selected in the top pane, as well as other information.



You can double-click on a frame in the top pane to open a window for editing the object's source code. You can press the option key and then double-click to open an inspector window on a frame's function. You can double-click on a line in the bottom pane to open an inspector window for the selected parameter.

Stack window commands

The Stack window has seven commands: Show Home, Inspect Method, Copy Stack as Text, Compute More Frames, Compute All Frames, and Hide/Show Internal Frames.

Show Home

Expands and highlights the home of the selected object. This is the same as the Show Home on the Browse menu. See Show Home for more information.

Key shortcut: **Command-Y**, or double-click the stack frame.

Inspect Method

Inspects a method in the top frame.

Copy Stack as Text

Copies the stack as text into the Clipboard so you can print out a copy of the top frame. Shows only the names of stack frames, not local variables.

Compute More Frames

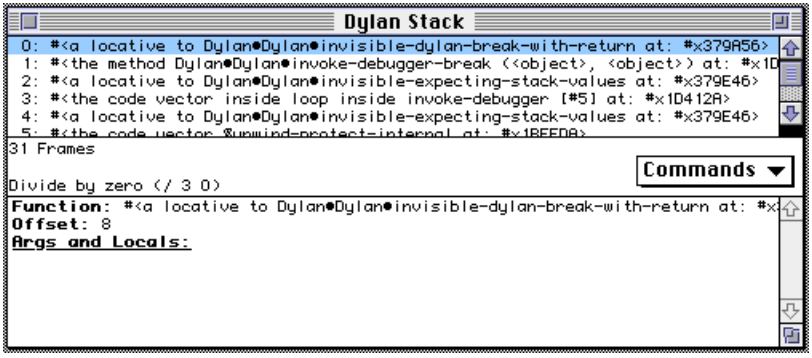
Recomputes the selected frames.

Compute all Frames

Recomputes all the frames.

Hide/Show Internal Frames

The internal frames are listed in the top pane. The internal frames are generally uninteresting to most programmers and should be set to Hide Internal Frames.



Inspector window commands

All inspector windows display the selected object and its class. The commands on inspector windows are Resample, Edit, and a list of commands specific to the object being inspected. The complete list of these commands include Show/Hide Slots, Who Uses Object, Show/Hide Disassembly, Edit Definition, Graph Class, Inspect General Instances, Inspect Direct Instances, and Show/Hide Elements.

You can open an inspector window by double-clicking on an object in an inspector window. If you have several inspector windows open at once, you can hold down the Option key while closing one inspector window and all of them will close.

Resample

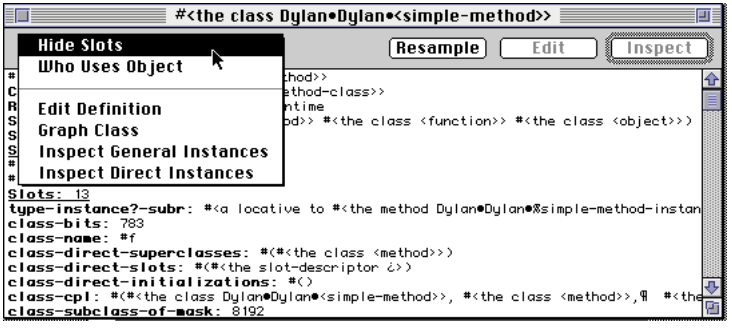
Resamples the object in case it has changed. The development environment caches data when it opens an inspector. If the object changes, the development environment still shows the cached data rather than the data on the runtime. Resample forces the development environment to retrieve the new data from the runtime.

Edit

Opens a window where you can edit the text of the object selected in an inspector window.

Show/Hide Slots

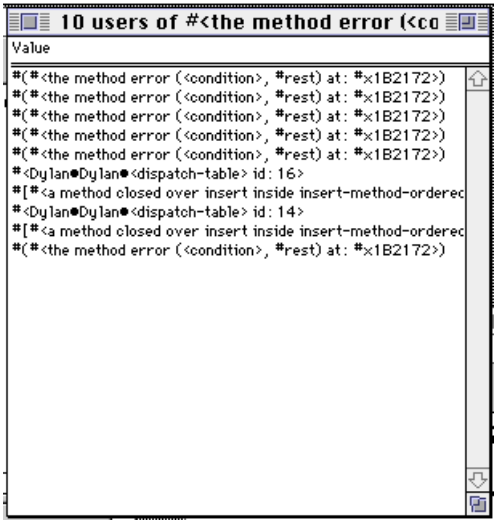
Hides and displays the slots of the object shown in the top line of the inspector. In the following figure the slots are on display and the command to hide them is about to be chosen.



Who Uses Object

Opens a windows that lists all the objects that contain a direct reference to the object. The reference can be a slot that refers to the object or a collection that contains the object.

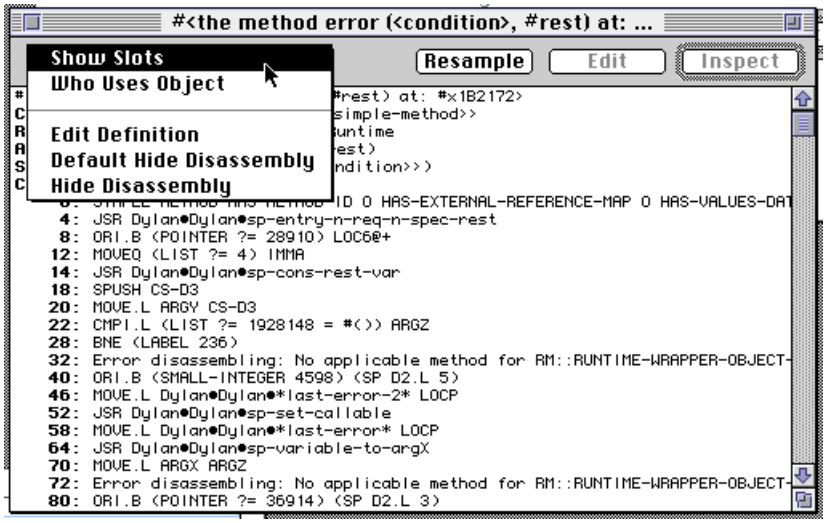
The following window is a typical result of Who Uses Object. In this example, the method error (<condition>…) was selected in the inspector window and the Who Uses Object was issued.



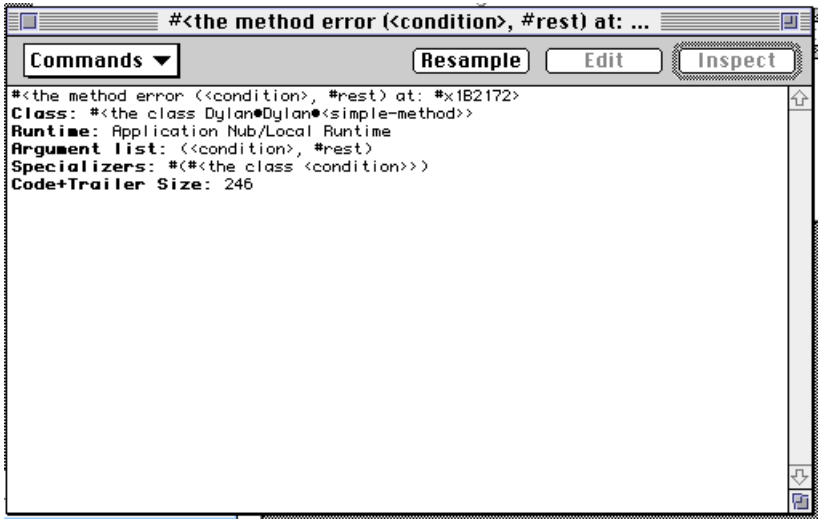
Show/Hide Disassembly

Hides or displays the machine instructions compiled for the inspected function.

The following figure shows disassembly code.



The following figure shows the disassembly code hidden.



Edit Definition

Opens a browser displaying the source code of the function or the class, whichever is selected.

Graph Class

Opens a browser displaying a graphical representation of the selected class.

Inspect General Instances

Opens the inspector window Inspect Object of Class, which lists all the objects whose type is that of the selected class or a subclass of it.

Inspect Direct Instances

Opens the inspector window Inspect Object of Class, which lists all the objects whose type is that of the selected class.

Show/Hide Elements

Hides or displays the elements of a collection.

Launch Application Nub — Project

Launches the Application Nub, loads the active project into it, and tethers the Application Nub to the development environment. When you are tethered to the development environment, the Apple Dylan Listener is usable and your project can be run and debugged. The type of target chosen using the Target Architecture command is the type of Application Nub launched with this command.

Key shortcut: **Command-K**

Related commands: Tether to Application, Quit Application Nub, Run.

When you tether to the development environment, you can see that the Application Nub is running by leaving Apple Dylan and checking the list of running applications in the Finder. If you switch to the Application Nub from

that list, you see that no windows or menus appear. You can return to the development environment by clicking in one of its windows.

The notation "(Unconnected)" disappears from the Listener when the Application Nub is tethered to the development environment.

See the Environment sheet under Preferences for whether to launch to the Application Nub automatically when a project is opened. Using Preferences, you can also choose to automatically issue Update Project before launching the Application Nub.

You can replace an Application Nub with another by removing the file Application Nub from your project's folders and replacing it with another Application Nub. The development environment searches first in the project folder and then in the folder Application Nub. You can also use an alias to another Application Nub as the replacement.

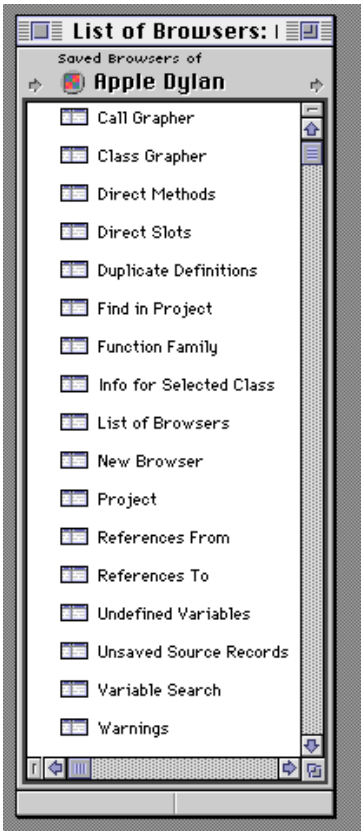
Listener

See Apple Dylan Listener.

List of Browsers Browse

Provides a drag and drop interface to the browsers available in the development environment. From this list, you can:

- Drag and drop a browser onto a pane splitter and have that browser embedded into any other browser.
- Drag and drop a browser into the inbox of a pane to replace the pane with that browser.
- Open a browser by double-clicking on it. This is the same as selecting the browser on the Browse menu.
- Drop an object onto a browser name to open that browser with that object as the basis. This is the same as selecting an object and then selecting a browser.



Most of the objects on this list are browsers listed on the Browse menu; they are the same browsers. This list also contains the command New Browser from the Browse menu. Find in Project and Project are not on the Browse menu. Find in Project opens a browser with the object you are looking for as its basis. The Project object is the Project browser for the active project, which is brought to the front when this object is selected. Browsers appear on this list because their files are in the subfolder _System Browsers of the Browsers folder in Apple Dylan.

You cannot delete a browser from this list. To delete a browser, quit the development environment, find the browser file in the Browser folder _System Browsers and move it out of the Browsers folder entirely. When you restart the development environment, the moved browser will no longer be listed.

Load UI Builder — Project

Loads the user interface builder into the Application Nub. This causes the Show Interface Builder command to appear on the Apple menu, which runs the Apple Dylan interface builder from within the development environment. You should launch the Application Nub before using this command. Note that you can run the interface builder from the Finder by double-clicking the standalone version of the interface builder.

For more information on the interface builder, see the book *Creating a User Interface in Apple Dylan*.

Related commands: Show Interface Builder, Hide Interface Builder.

Look Up in Online Reference — Browse

Looks for the online documentation reference entry that pertains to the selected text and displays it with the QuickView application. Searches the Macintosh Programmer's Toolbox Assistant, or in the Apple Dylan language constructs.

Key shortcut: **Command-=**

The selected text can be text inside a source record or the Listener, such as Inside Macintosh routines, header files, framework functions, or classes, Creole extensions, or Dylan language constructs. You can use Look Up in Online Reference even if you have not purchased the Macintosh Programmer's Toolbox Assistant and installed it on your computer, but in that case you won't be able to search for Macintosh toolbox calls. If the entry exists, QuickView displays the selected reference page. If the entry does not exist, no page appears on your screen.

QuickView runs as a separate application from Apple Dylan. You can quit QuickView and continue running Apple Dylan.

The Macintosh Programmer's Toolbox Assistant is available on a number of different CD-ROMs from Apple:

- Macintosh Programmer's Toolbox Assistant CD-ROM $89.95 from APDA (part number T1616LL/A) or a bookstore near you.
- The Developer CD Series (March '95 and later Reference Library Editions) 12 CD-ROMs a year for $250.
- E.T.O. #17 and later—about $1,100

- Some versions of MPW Pro—about $500

Meter Expression — Debug

Meters an expression in the runtime. The Meter Expression window allows you to type or copy in the expression you want to meter. When you run the expression, the total time used and bytes alloted are calculated. In addition, the generic functions called and classes allocated during the execution of the expression appear. You must select the module the expression is in before running it.

You can sort either of the lower panes in the Meter Expression window by clicking on the column's name you want the pane sorted by. The name of the column that controls the sort is then underlined.

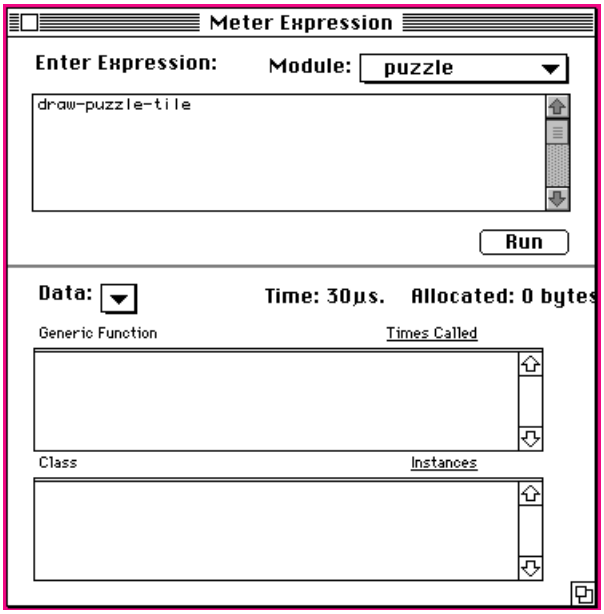
Meter Expression shows basic information about almost all the generic functions and classes in a project. For detailed information about specific methods or generic functions, use Call Recording.

Related command: Call Recording.

The total time used and bytes allocated are calculated in the Meter Expression window. The first time you call a function it sometimes runs more slowly than it will normally. If the generic function `Dylan.Dylan.finalize` appears on the list, this is the first time an instance of this class has been created. You should run the expression again for accurate timing.

If the generic function `set-generic-function-dispatch<<Dylan<<Dylan` appears on the list, it indicates that this is the first time one of the generic functions has been called. Sometimes if you add a method to a generic function, it needs to be treated as if it is being called for the first time. You should run the expression again to get more accurate timing results.

The following figure shows the total time used and bytes allocated for `draw-puzzle-tile`, which is in the puzzle module.



Generic function calls that have been inlined are not displayed in the Generic Function pane. You can double-click on a generic function to open an inspector window on the generic function.

Some built-in classes, such as `<list>` and `<byte-string>`, are not displayed in the Class pane but are counted in the Allocated bytes field. You can double-click on a class in the Class pane to open an inspector window on the class. The number of instances of the class are listed in the inspector window.

See Inspect Stack for more information on inspector windows.

Method Template (Copy Special) — Edit

Copies a prototype method template for the selected generic function or method from the active project into the Clipboard. To paste the method template into a source record or the Listener, use Paste. You can select either its source record icon or text naming the function or method.

Key shortcut: **Command-T**

Method Template (Copy Special) helps you create new methods, especially if you are specializing the arguments of a method. For example, if you select the method `floor`, Method Template (Special Copy) copies the following template into the Clipboard:

```
define method floor(numerator :: <object>, denominator:: <object>)
                          => quotient :: <integer>, remainder :: <real>

end method floor;
```

Method Template (Copy Special) retrieves the prototype argument list or template for the selected object from the compiler results database and inserts it into a source record. You can then specialize the types of the arguments and fill in the body of the method.

Note that Method Template (Copy Special) appears in the menu only if you select a generic function or method. If you select a class, Class Template (Copy Special) appears instead.

Related Commands: Copy Special, Method Template (Insert Special), Class Template (Copy Special).

Method Template (Insert Special) Edit

Replaces the selected generic function or method with the prototype template of the methods for the selected method. Method Template appears in the menu only if you select from the active project editable text that names a function or method. This command is made available by pressing Option when clicking on Copy Special on the Edit menu.

For example, if you select the function `draw` and hold down the Option key, Method Template (Insert Special) replaces `draw` with the following text:

```
define method draw(view, r)

end method;
```

Key shortcut: **Command-Option-T**

If you select a class Class Template (Insert Special) appears in the menu instead of Method Template (Insert Special).

Method Template (Insert Special) retrieves the prototype argument list or template for the selected object from the compiler results database and inserts it into a source record.

Related Commands: Insert Special, Method Template (Copy Special), Class Template (Insert Special).

Modules of (Aspect) — Browse

Operates on the active project, listing its modules.

Modules of (Aspect) is commonly used to customize a browser pane displaying that aspect of a class. See Aspect and "Customizing browsers" on page 58 for more information.

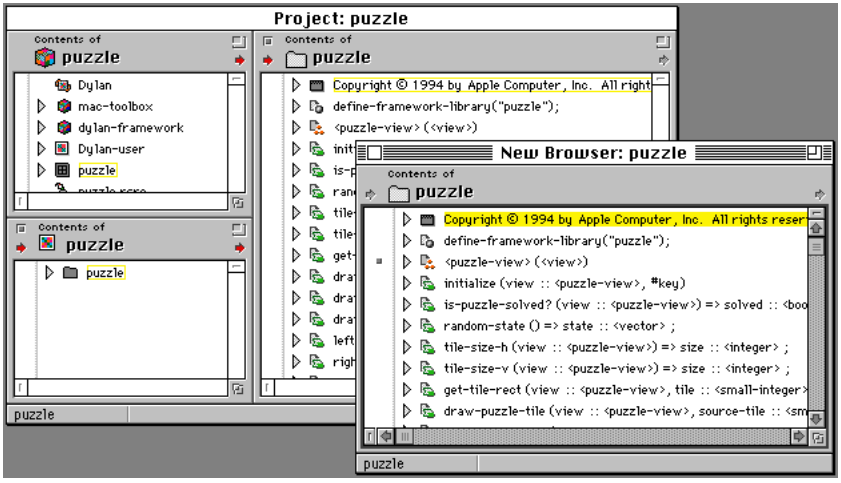
New Browser — Browse

Opens a new browser with a single pane. The new browser's pane contains the default aspect of the object selected prior to choosing New Browser. The selected object is called the basis of the browser's pane and, therefore, the basis of the browser. If no object is selected prior to choosing New Browser, the new browser is empty.

For more information on using browsers, see the section "Using browsers" on page 15.

You can also create a new browser by double-clicking on an object.

The following figure is a new browser that was created with the source folder "puzzle" selected.



If you want to have more than one pane in a browser, you split the pane in two. Each of the two new panes can also be split, both horizontally and vertically. You split a pane with its splitter controls; the horizontal splitter control is a short, horizontal bar just above the scroll bar on the right of a pane. The vertical splitter control is a short, vertical bar just to the left of the scroll bar at the bottom on a pane. To split a pane, hold down the mouse on the bar and then drag the dashed line that appears to wherever you want the pane split. Another method is to drag the outbox of a pane onto any splitter on any pane. This splits the pane you drop it in and creates a link to the new pane.

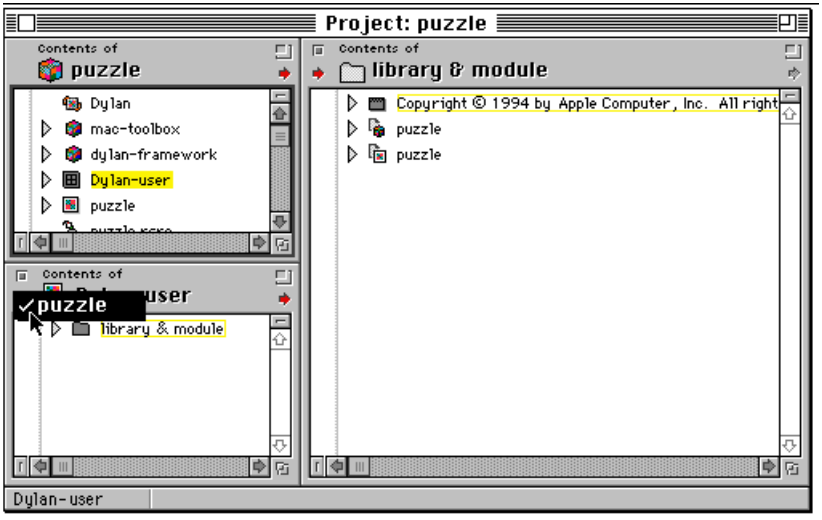
You can close a pane by clicking its close box in the upper-left corner of its header. The browser remains open as long as you don't close the only pane in it. You can resize a pane by dragging its resizing control in its lower-right corner.

You can change the relative size of panes by dragging the line that separates two panes. You move the cursor to the separating line until it changes into a slider cursor, then you hold down the mouse and slide it to where you want the new separating line to be. For more information on splitting panes and resizing them, see the task "Customizing browsers" on page 58.

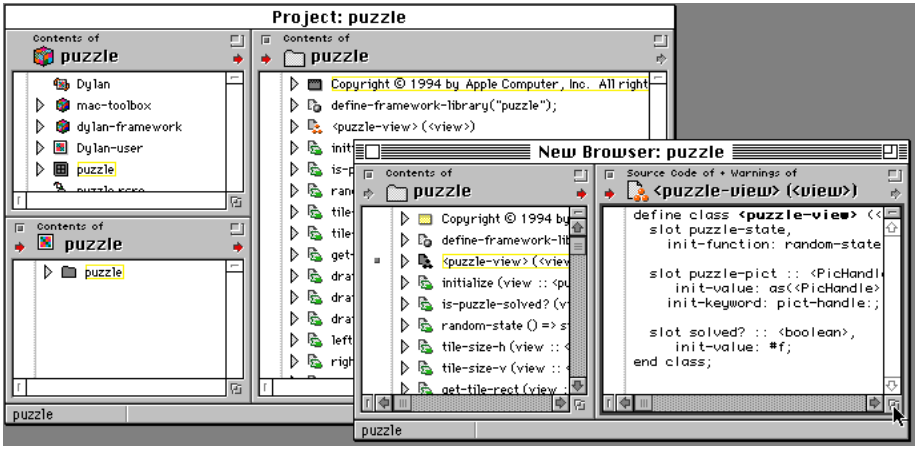
The panes in a browser work together through links. The contents of an object you select in one pane shows up in another through a link you can create or change. To see a link's origin or destination, hold the cursor over the link's

inbox or outbox (the small arrows at the edges of the header of each pane). When a link has been established, the arrow turns red.

In the following figure the origin of the incoming link of the lower-left pane is revealed by holding down the mouse on its inbox. The origin of this incoming link is root pane for the puzzle project, which is the upper-left pane.



When you split a pane, the original and the new panes are automatically linked. In the following figure the split is complete, so the outbox of the original pane on the left and the inbox of the new pane on the right are red. You can now click on objects in the left pane and they are displayed in the right pane. Because <puzzle-view> was selected in the original pane, it is displayed in the new pane.



For more information on using links, see the task “Linking panes and browsers” on page 34.

Related Command: Save Browser.

New Module File

Creates a new module named Untitled with a source folder named Untitled in it. A second new module is named Untitled 2 with a folder named Untitled 2 and so forth. These source folders each contain an empty source record.

You can rename modules either by clicking on their names, or by selecting them and pressing the Return key.

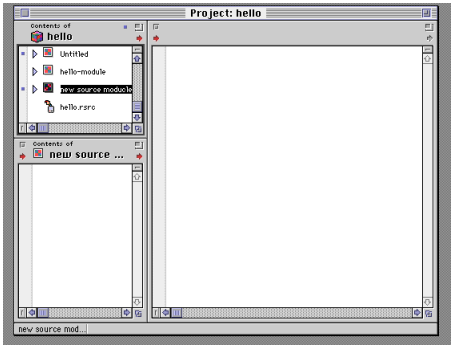
Note

The New Module command identifies a module to Apple Dylan, but you must also identify the module to the compile. To do this, you must create a source record with a `define module` statement or add a `use` statement to the current library defined with `define library`. See “What goes into a project?” on page 97.

New modules are placed at the end of the list of modules. You can reorder the modules by dragging them. Changing the module order changes the load order. In the Dylan language, forward declarations do not create load order

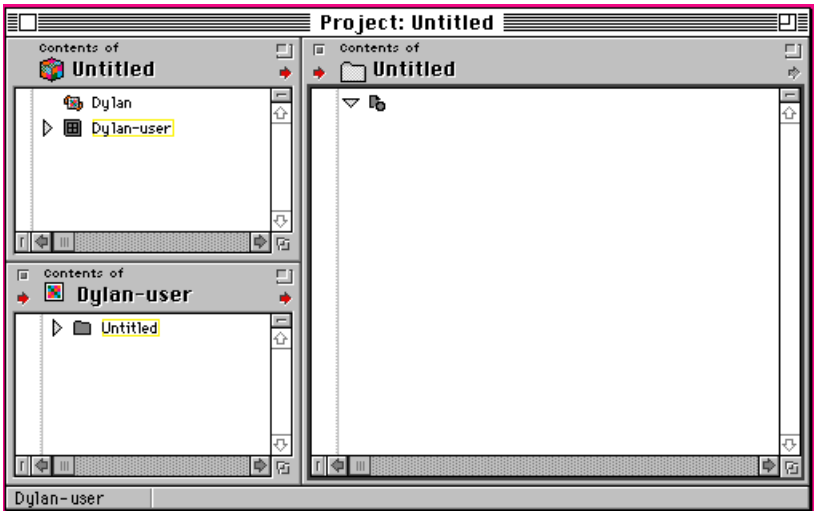
restrictions. However, load order dependencies, such as calculations that are performed in one module based on a variable previously defined in another module, must be accounted for in the module order you establish. For more information on load order, see the *Dylan Reference Manual*.

The following figure shows the new module at the end of the list of modules.



New Project — File

Creates a new project and opens a project browser for it. If no other project is already open, the new project becomes the active project. Otherwise, the new project is simply opened. The following figure shows the default project browser for a new project.



New Source Folder File

Creates a new source folder named Untitled with an empty source record in it.

You can rename source folders either by clicking on their names or by selecting them and pressing the Return key.

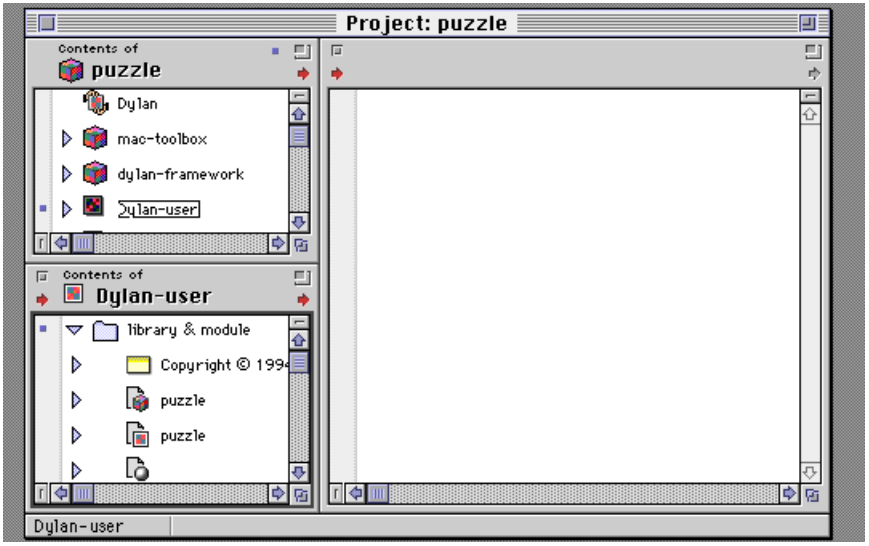
New source folders are placed at the end of the existing list of objects by default. If a source folder is selected, the new source folder is placed directly after it. Because the order of the source records within a source folder and the source folders within a module determines the load order, you might need to move an object into its proper place after it's created. You can reorder the objects by dragging them.

New Source Record File

Creates a new, empty source record. The empty source record has a generic icon until its contents are completed. Source record objects include methods, classes, variables, constants, macros, generic functions, comments, and top level forms.

Key shortcut: **Command-N**

The following figure shows a new source record that has been created for the puzzle module. The new record appears at the end of the list of other records by default. If you place the cursor into the name of an existing source record, the new source record is placed directly after it instead of at the end. Because the order of the source records within a source folder and the source folders within a module indicates the load order, you might need to move an object into its proper place after it's created. You can reorder the objects by dragging them.



New Text Window File

Opens a new text window. Use this window to make READ ME files, release notes, scratch pads, and so forth. All the usual Apple Dylan editing commands are supported. You are prompted for a file name and destination when you close the window. The default output file name is "new.dylan".

Use Export and Import for moving text files in and out of Apple Dylan.

Open File

Opens a project. You can also open a project by double-clicking on the project's icon from outside Apple Dylan. This launches Apple Dylan if it is not already running.

Key shortcut: **Command-O**

You can have several projects open at once, but the first one you open is the active project. The active project is the only project that can be browsed, compiled, and downloaded.

You can also open text files with this command.

Page Setup File

Displays the standard Macintosh Page Setup dialog box to allow you to determine the behavior of the Print command.

Related command: Print.

Paste Edit

Pastes text from the Clipboard into the current text selection or at the current cursor location. Text is placed into the Clipboard using Cut and Copy, as well as by all three of the Copy Special commands.

Key shortcut: **Command-V**

Related commands: Copy, Copy Title Text, Cut, Copy Special.

See "Editing in Apple Dylan" on page 66.

▲ **WARNING**
Emacs-style editing commands create a kill ring that may interact unpredictably with the Clipboard. See "Editing in Apple Dylan" on page 66.

Pattern Match Macro — Debug

Prints the macro expansion first, displaying exactly the text sent to the compiler by the parser. Then prints the pattern rule, showing which elements of the macro match which elements of the pattern. From this information you can determine if there is a match or not. If there is a match, you can determine its scope. You can also determine whether the order of pattern matching is correct, catching such errors as matching a general pattern before matching a specific pattern (which would never be seen). This process is repeated for all further macro expansion.

Operates on statement macros and definition macros. Does not work on function macros. This command does not work on the icon; you must select the full text of the macro in a browser or the Listener. Expands the selected macro to the Listener. You can also select the macro text in the Listener.

This command is used in debugging macros.

If you select Pattern Match Macro from the menu while pressing the Option key, the Pattern Match Macro Including Builtin command appears in the menu instead. The behavior is the same, except that all macros are expanded, including builtin macros from the Dylan language, such as `case` or `begin`.

Related command: Expand Macro, Pattern Match Macro Including Builtin.

Pattern Match Macro Including Builtin — Debug

Prints the macro expansion first, displaying exactly the text sent to the compiler by the parser. Then prints the pattern rule, showing which elements of the macro match which elements of the pattern.

You must select Pattern Match Macro while pressing the Option key to make the Pattern Match Macro Including Builtin command appear on the menu. The behavior is the same as for the Pattern Match Macro command, except that all macros are expanded, including builtin macros from the Dylan language, such as `case` or `begin`.

This command is used in debugging macros.

Related commands: Expand Macro, Pattern Match Macro.

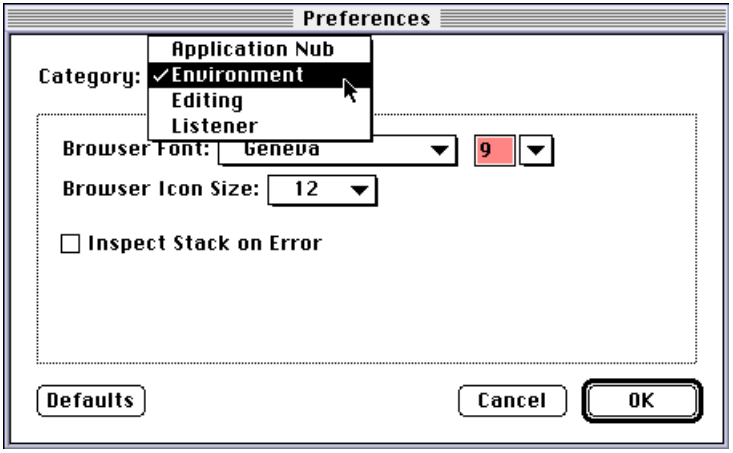
Preferences — Edit

Opens the Preferences dialog box, which consists of four sheets:

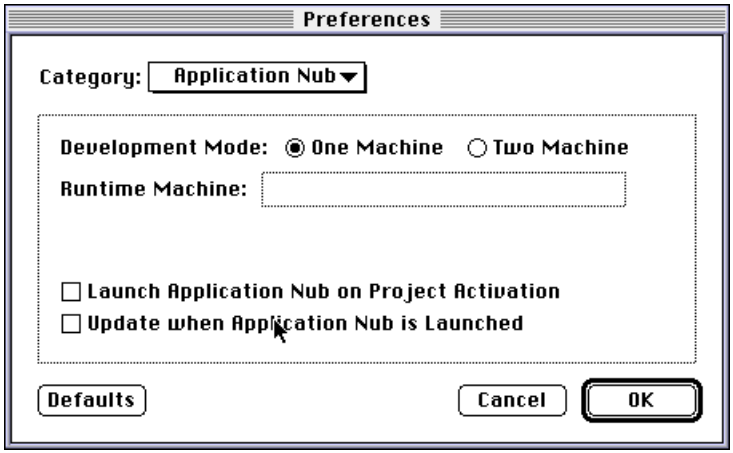
- Application Nub
- Environment
- Editing
- Listener

By setting preferences on these sheets, you can make the development environment interact with each of your projects in a specified way.

The following figure shows the four sheets on the Category pull-down list. Clicking any Defaults button resets all the values on all three sheets to their defaults.

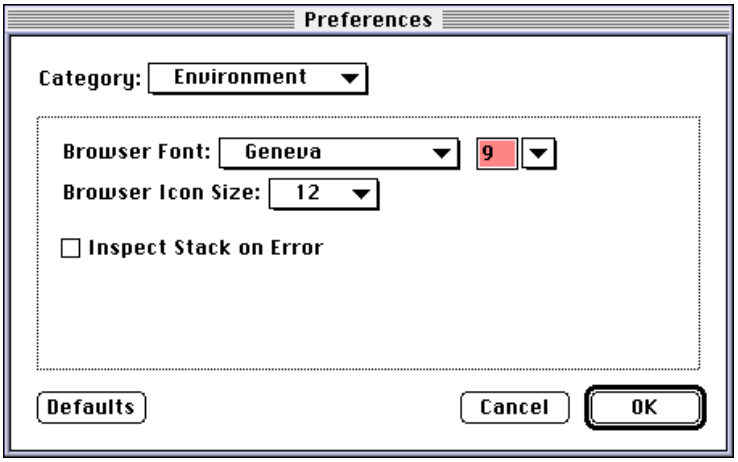


The following figure shows the default Application Nub Environment sheet.



- Development Mode—One-machine development is the default. Choose two-machine development if you don't want your runtime on the same machine with the development environment.
- Runtime Machine—For two-machine development, name the machine the runtime is on.
- Launch Application Nub when Active Project is Opened—Off by default. See "The Application Nub" on page 107, and Launch Application Nub.
- Update when Application Nub is Launched—Off by default. See "Keeping your project synchronized" on page 109, and Update Project.

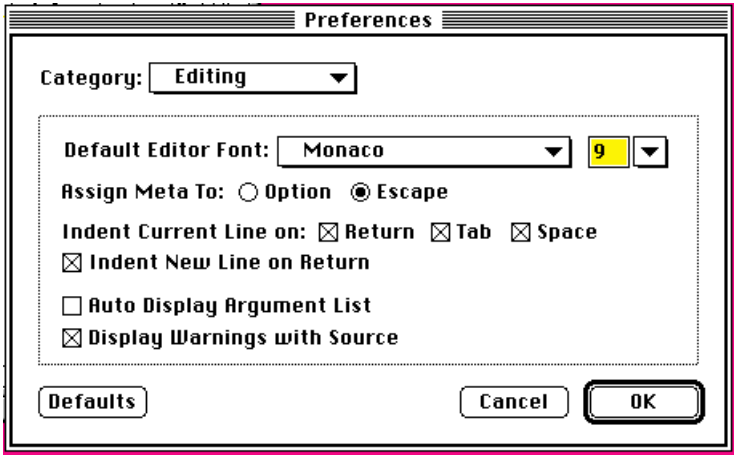
The following figure shows the default Environment sheet. Not all choices are reflected in the development environment until you open a new project.



Preferences on Environment sheet:

- Browser font and size—Not the source text font, but the title font you see when an object is collapsed.
- Browser icon size—The smaller the icons, the more your browser can display.
- Inspect Stack on Error—If you prefer, you can go directly to the stack inspector on any error.

The following figure shows the default Editing sheet.

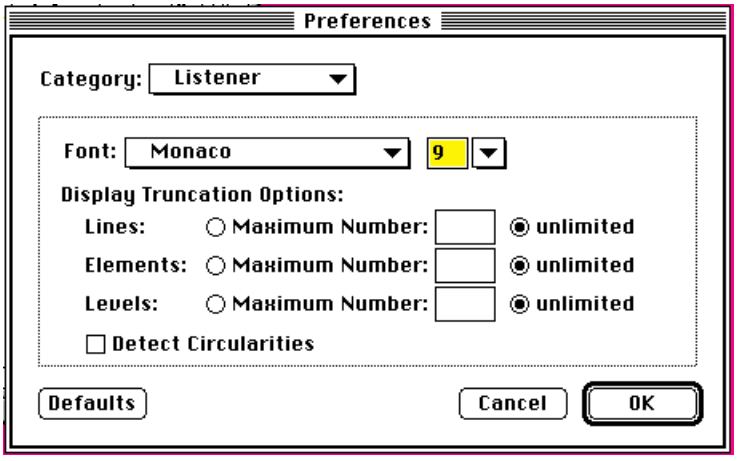


Choices are reflected in the text you write or edit, such as source code. Any code entered before this preference is set remains unaffected. Changing this font setting only affects new source text.

Preferences on the Editing sheet:

- Default editor font— the font for editing source code. The fonts used for code entered before setting this are not affected.
- Assign Meta To—assignment of meta key. See "Emacs-style key commands" on page 72.
- Indent Current Line on—part of auto-indent system. Choose how often you want the current line checked for "pretty" indentation.
- Indent New Line on Return—extends auto-indent to the next line.
- Auto Display Argument List—Automatically display the argument list.
- Display Warnings with Source—Automatically display warnings with source code.

The following figure shows the default Preferences sheet for the Listener.



Preferences on the Listener sheet:

- Default Listener font—the font for editing code in the Listener.
- Set truncation of lines—Unlimited by default; controls the number of lines printed in the Listener.
- Set number of elements—Unlimited by default; controls the number of elements of a list printed in the Listener.

- Set number of levels—Unlimited by default; controls the number of break levels displayed in the Listener.
- Detect circularities—Off by default. If set, stops printing after the first time around.

Print — File

Displays the standard Macintosh Print dialog box for printing. You can print the contents of the Apple Dylan Listener or source text from a browser. If nothing is selected, the entire project is printed.

Key shortcut: **Command-P**

Related command: Page Setup.

Quit — File

Quits the development environment. You are prompted to save unsaved changes and prompted to confirm that you wish to quit. Issuing this command also quits the Application Nub if you are tethered and performs an orderly shutdown of Apple Dylan.

Key shortcut: **Command-Q**

Quit Application — Project

Quits a running application and the Application Nub. Quit Application is identical in effect to Quit Application Nub.

Quit Application quits a running application and also untethers the Application Nub from the development environment (if connected). Quit Application is the opposite of Tether to Application.

Related command: Tether to Application.

Quit Application Nub — Project

Quits a running application and the Application Nub. Quit Application Nub is identical in effect to Quit Application.

Quit Application Nub quits a running application and also untethers the Application Nub from the development environment (if tethered). Quit Application Nub is the opposite of Launch Application Nub.

If you encounter difficulty quitting the nub, run the Quit Application Nub application from the Finder.

Related command: Launch Application Nub.

Recompile — Project

Initiates a complete compilation of the entire project, including, optionally, its subprojects. All code within the chosen scope is recompiled whether it needs it or not, regardless of status marking. Recompile throws away all caches and cleans up all databases.

▲ **WARNING**
Recompile is of limited value when tethered because it adds all results of its recompilation to the existing compiler results database and removes nothing.

Related commands: Compile Expression, Compile Region, Compile Selection.

See also: Create Application, Create Library.

Redo Clear — Edit

Deletes text that you had previously reinstated with an Undo Clear. The deleted text does not get copied to the Clipboard. Undo, Clear, Redo Clear, and Undo Clear replace one another on the File menu, depending on which is appropriate.

See “Editing in Apple Dylan” on page 66.

▲ **WARNING**
Clear supports Undo and Redo only for text. If you have cleared an object, folder, or subproject, it cannot be undone or redone. If you are about to clear a subproject, you are warned. You cannot clear required subprojects.

References From — Browse

Operates on the selected class, method, or variable, opening a browser that lists all objects from which it is referenced.

Related commands: References From (Aspect), References To.

References From (Aspect) — Browse

Operates on the selected class in the active project, method, or variable, listing all references from it.

References From (Aspect) is commonly used to customize a browser pane displaying that aspect of a class. You can also select a single class in a pane and display references from the selected object inline.

See Aspect and "Customizing browsers" on page 58 for more information.

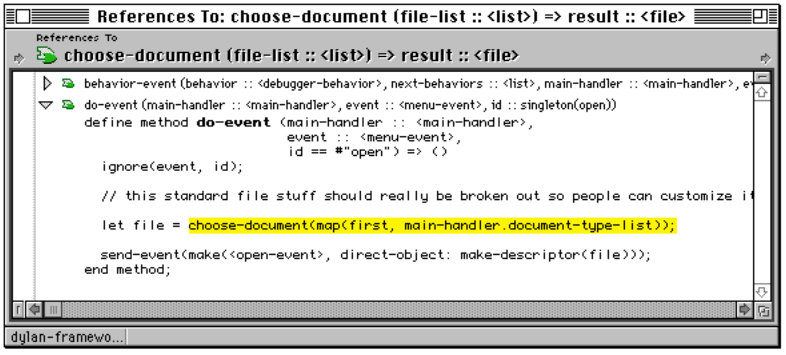
The References From command opens a single-pane browser displaying references from the selected object.

References To — Browse

Operates on the selected class in the active project, method, or variable, opening a browser that lists all objects making reference to it.

See "Using the browser References To" on page 49.

Related commands: References To (Aspect), References From.



References To (Aspect) — Browse

Operates on the selected method, class, or variable in the active project, listing all references to it.

References To (Aspect) is commonly used to customize a browser pane displaying that aspect of a method, class or variable. You can also select a single object in a pane and display references to it inline.

See Aspect, "Using the browser References To" on page 49, and "Customizing browsers" on page 58 for more information.

The References From command opens a single-pane browser displaying direct methods.

Replace and Find — Text

Duplicates Replace & Find in the Find/Replace dialog box. Replaces the target text with the replacement text and searches for the next instance of the target text.

Key shortcut: **Command-J**

Related command: Find/Replace.

See "Editing in Apple Dylan" on page 66.

Reset Stack — Debug

Discards the stack all the way to the top of the stack without running any of the clean-up code. Because it does not run any clean-up code, Reset Stack is very risky; try Abort first. Severe side effects can result from not running the clean-up code. For example, a file can be left open with no way to close it, or a data structure, such as the Dylan subproject itself, can be left in a corrupted state.

Reset Stack does not discard the runtime heap. To reset the heap, untether the Application Nub from the development environment and then retether to it.

See "Debugging a project" on page 146 for a discussion of the choice between Abort and Reset Stack.

Related command: Abort.

Resource Files of (Aspect) — Browse

Operates on the active project and its subprojects, listing their resource files.

Resource Files of (Aspect) is commonly used to customize a browser pane displaying that aspect of a project.

See Aspect and "Customizing browsers" on page 58 for more information.

Revert — File

Reverts to the saved version. This command works on either the whole project or on selected objects. If you select more than one object, the next higher container and all its contents are reverted. All changes you made since issuing the last Save or Save All are lost. Revert cannot be undone.

Run — Project

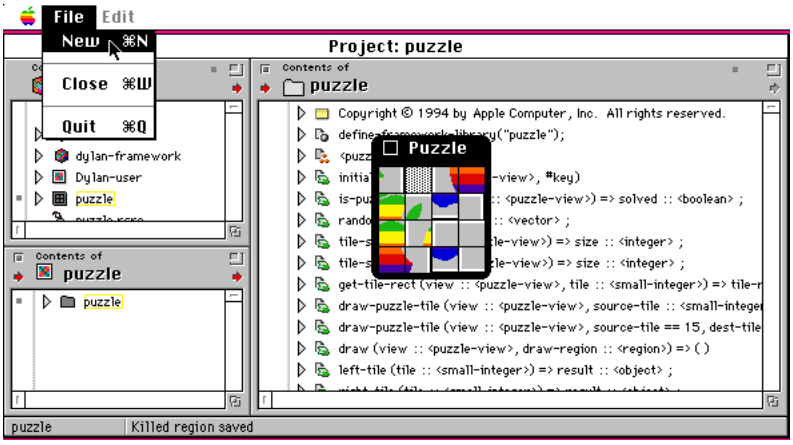
Runs the active project's startup function after tethering the Application Nub to the development environment and performing an update. Run performs different functions depending on the state of your project, but it does compile all sources marked as uncompiled and downloads the results, just as Update Project does. It also tethers to the development environment, if needed, and runs the startup function.

You can designate the active project using Set Project Type. The name and location of the project's startup function must have been specified using Set Project Type or Run will not work.

Key shortcut: **Command-R**

When you run a project from within Apple Dylan, the user interface for the application and its menu bar overlay the development environment. The application's menu bar can be used to manipulate the application and to quit it. You can return to Apple Dylan by clicking in one of its windows or choosing Application Nub from the Finder's list. If you leave the application running and return to Apple Dylan, the cursor becomes the watch cursor to remind you that the application is still running, even though you can no longer see it.

Note
You cannot use Apple Dylan again until you interrupt your running application. Use Quit within your application to do this.



Related commands: Launch Application Nub, Quit Application, Update Project.

Save — File

Saves the sources of the active pane and all the objects it contains. Save cannot be undone.

Key shortcut: **Command-S**

Related commands: Save All, Revert.

For saving a new or revised browser configuration, see Save Browser.

Save All — File

Saves the sources for all open projects. Save All cannot be undone.

Related commands: Save, Revert.

For saving new or revised browser configurations, see Save Browser.

Save Browser — Browse

Saves the configuration of a browser you have created or changed. You cannot save the contents in the browser with Save Browser unless the basis of the browser is a module. Any incoming links or outgoing links to the browser are lost when it is saved.

You can name the saved browser what you want. Its name is added to the fourth section of the Browse menu and to the List of Browsers browser, if you save the file to the Browsers folder of Apple Dylan. If you save the file to the Browsers subfolder _Ignore Selection Browsers, the new browser appears in the third section of the Browse menu and the entire project is the only valid basis for the browser. If you save the file to the Browsers subfolder _System Browsers, the new browser appears only on the List of Browsers browser.

You can move browsers between subfolders, if you want. You can share a saved browser with others by giving them a copy of the browser file or an alias to it.

You can delete a saved browser from the development environment by removing its file from the Browsers folder. You must restart the development environment for the browser to be removed from the List of Browsers browser and the Browse menu.

Related command: List of Browsers.

See “Saving a browser configuration” on page 61.

Select All — Edit

Selects all the objects in the active pane or all the text in an object. To select all the objects in a pane, click on one of the objects and then choose Select All. To select all the text in an object, expand the object, click anywhere in the text and then choose Select All. You can also select all the text in an object by clicking in it four times.

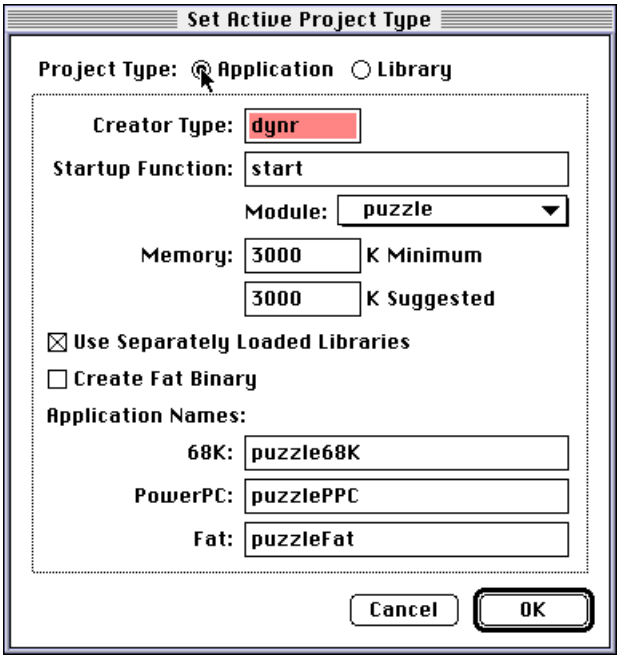
Key shortcut: **Command-A**

Set Project Type — Project

Sets the basic characteristics of a project, most importantly identifying its name and whether it is to be built as an application or a library.

You must set certain characteristics of a project before building your final application or library. After setting a project's features, you complete the building process by untethering from the development environment and using Create Application or Create Library. It is also recommended that you issue a Recompile command before building your final application or library.

The most important setting is whether the project is an application or library, as shown in the following figure.

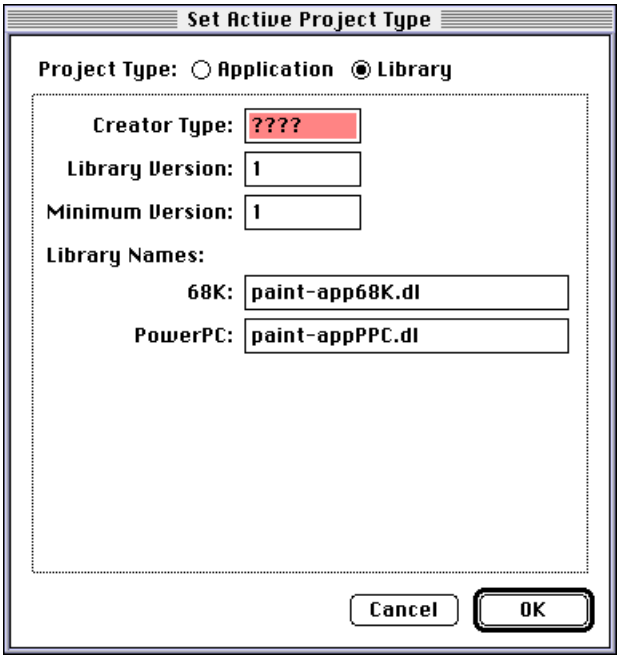


Specify the startup function by entering it in the Startup Function field. Specify the module your startup function is in with the Module field; this module must be exported from its library. The Creator Type field holds the creator ID you want your new application to have. Designate your memory requirements for suggested and maximum sizes in the spaces provided.

Check "Create Fat Binary" if you want to build a Fat application.

Check "Use separately loaded libraries" if you want the Dylan subproject and framework subproject not to be bundled with your application. They will not be included in your project's link library in the binary, but instead the link

library is loaded in from the extensions folder when the application is run. This option saves disk space, but not memory. This setting also allows you to revise and load new libraries for an application without changing the rest of the application.



Setting the version of a library controls which version is recorded in the compiler results database. When a library file is created from a project, the library versions of all of its subprojects are recorded in the library file header. Then at runtime, when the library is loaded, it will only accept those versions of its the sublibraries which are compatible with the version of the sublibrary used at library creation time. You should increment the version number whenever you release a new version of a library.

The minimum compatible version is the lowest version that a library is backward compatible with. The minimum version is checked during runtime library search. You should increment the minimum version number whenever you make incompatible changes. See “Library version numbers” on page 106 for a detailed explanation.

The suffix “.dl” is appended to the library’s name when you build it.

See “Apple Dylan User Model” on page 95, in particular “Application or library?” on page 104.

Show Home — Browse

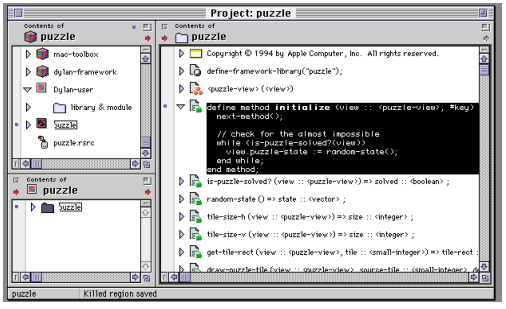
Displays and highlights the source code for the selected object. Checks first in the current module and goes immediately to the definition. If there’s no such name defined in the current module, searches all modules and goes immediately to the definition. If there are multiple definitions with the same name in different modules, presents a choice. If the source code is not found, displays the name of the object it was looking for.

Note
If there are multiple objects with the same name, and one is defined in the current module, you are not presented with a choice, but rather go immediately to the definition.

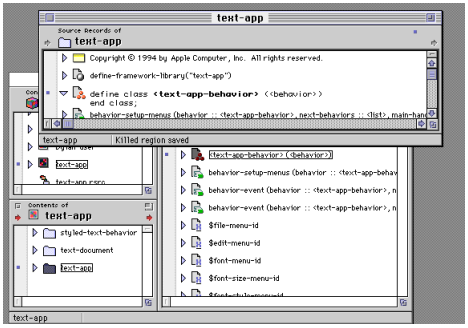
Key shortcut: **Command-Y**

For example, if you are in the Listener and have not been working with a certain function, you might not have its source record on display. In that case, it is easy to find the source code using Show Home. To use Show Home, place the cursor in the name of the function you are interested in, or select it, and then issue Show Home.

In the following figure you can see the result of clicking on “initialize” in the Listener and then choosing Show Home; “initialize” is expanded and highlighted.



The following figure shows the result of clicking on the “text-app” module in the root pane and then choosing Show Home; a new browser is opened with text-app as its basis.



Show Interface Builder — Apple

Runs the Apple Dylan interface builder from within the development environment. Note that you can run the interface builder from the Finder by double-clicking the standalone version of the interface builder.

For more information on the interface builder, see the book *Creating a User Interface in Apple Dylan*.

Related commands: Hide Interface Builder, Load UI Builder.

Show Interface Mapping — Debug

Prints information about the mapping between C and Dylan names to the Listener. Choose this command to set this feature, which prints the information whenever a define interface statement is compiled. This command allows you to check name mapping, plus import and export options. To disable this feature, choose the command again to remove the check mark from the menu.

When this command is checked, the following is printed at the Listener prompt: the C name, which is followed by an arrow, and the Dylan name.

As an example, when the following definition is compiled:

```
define interface
  #include "Dialogs.h",
```

```
    name-mapper: identity-name-mapping,
    define: {"SystemSevenOrLater"},          // for Gestalt
    CFM-library: "InterfaceLib",
    import: {"ErrorSound", "Alert", "ParamText"},
    // These next two will be null pointers anyway
    type: {"SoundUPP" =3D> <machine-pointer>},
    type: {"ModalFilterUPP" =3D> <machine-pointer>};
  #include "Resources.h",
    name-mapper: identity-name-mapping,
    define: {"SystemSevenOrLater"},          // for Gestalt
    CFM-library: "InterfaceLib",
    import: {"OpenResFile"};
end interface;
```

The following output is printed in the Listener:

```
function ErrorSound => ErrorSound<<Online-Insultant
function ParamText => ParamText<<Online-Insultant
function Alert => Alert<<Online-Insultant
function OpenResFile => OpenResFile<<Online-Insultant
```

Size — Text

Displays a list of font sizes. The size you choose is applied to the selected text.

This choice changes only the text in a source record and has no semantic impact. but it does flag the record for recompilation. You can use font size to help organize your sources.

See "Formatting commands" on page 70.

You can change the font sizes for the Listener and Browsers (as well as the source font size) using Preferences.

Source Code of (Aspect) — Browse

Operates on source records and definition entities in the active project, displaying the definition associated with the selected source record or

definition entity. Source record objects are methods, classes, variables, constants, macros, generic functions, comments, and top-level forms.

Source Code of (Aspect) is commonly used to customize a browser pane displaying that aspect of a module. You can also select a single module in a pane and display its source code inline. See Aspect and "Customizing browsers" on page 58 for more information.

Source Folders (Aspect) Browse

Operates on modules in the active project, listing the source folders for the selected object.

Source Folders (Aspect) is commonly used to customize a browser pane displaying that aspect of a module. You can also select a single module in a pane and display all its source folders inline. See Aspect and "Customizing browsers" on page 58 for more information.

Source Records (Aspect) Browse

Operates on any object in the active project that has source records: project, subproject, module, source folder, listing the source records for the selected object. Source record objects are methods, classes, variables, constants, macros, generic functions, comments, and top-level forms.

Source Records (Aspect) is commonly used to customize a browser pane displaying that aspect of an object. You can also select a single object in a pane and display its source records inline. See Aspect and "Customizing browsers" on page 58 for more information.

Source Records with Warnings (Aspect) Browse

Operates on any object in the active project that has source records: project, subproject, module, source folder, listing the source records for the selected object that have warnings in the active project. Source record objects are methods, classes, variables, constants, macros, generic functions, comments, and top level forms.

Source Records with Warnings (Aspect) is commonly used to customize a browser pane displaying that aspect of an object. You can also select a single

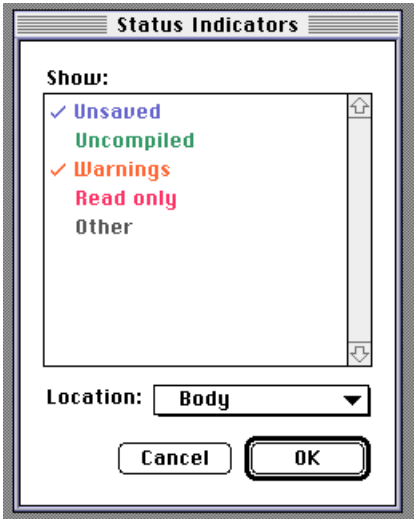
class in a pane and display its source records inline. See Aspect and “Customizing browsers” on page 58 for more information.

Status Indicators — Browse

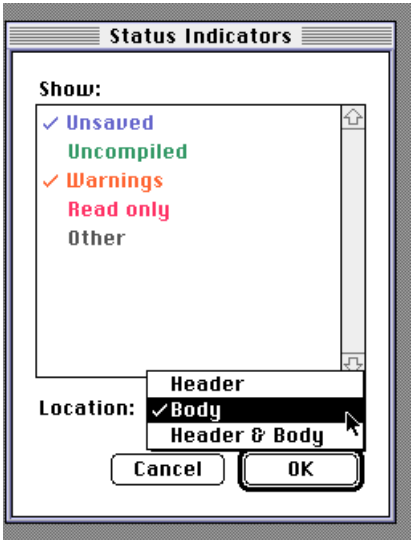
Opens the Status Indicators dialog box. You can also open this by double-clicking in the gray, vertical status indicator column in a pane. The status indicators you check in this dialog box are displayed in the status indicator column in the active pane or in the pane’s header. You select the status indicators separately for each pane.

To choose a status indicator, make the target pane active, open the Status Indicators window, and click to the left of the status indicator you want until a check mark appears. If you choose all the indicators, except Other, an indicator for each will appear in your pane when that status applies. The status indicator Other consolidates any of the unchecked indicators into a single indicator. By checking only the specific indicators you want to see, and also checking Other, you can see the specific indicators you need to see but still be informed that other status changes have occurred.

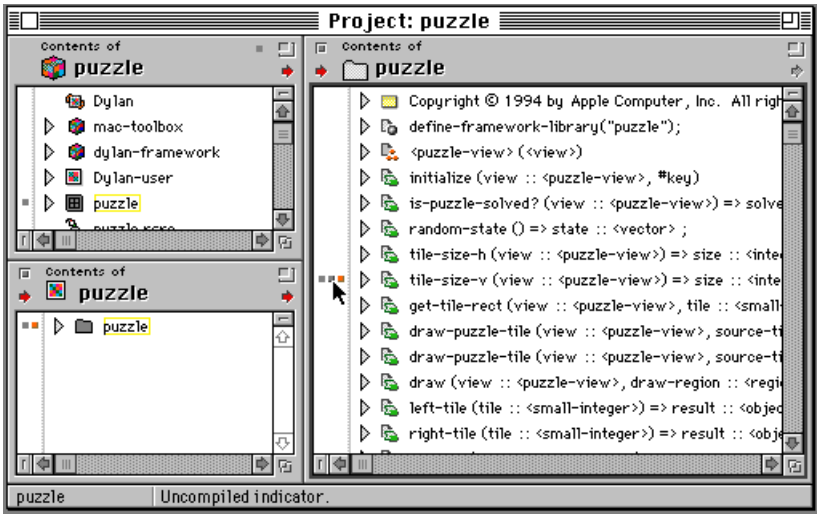
The following figure shows the default Status Indicators dialog box. Unsaved and Warnings indicators are displayed in the panes automatically. You can choose not to see them by unchecking them in the Status Indicators dialog box.



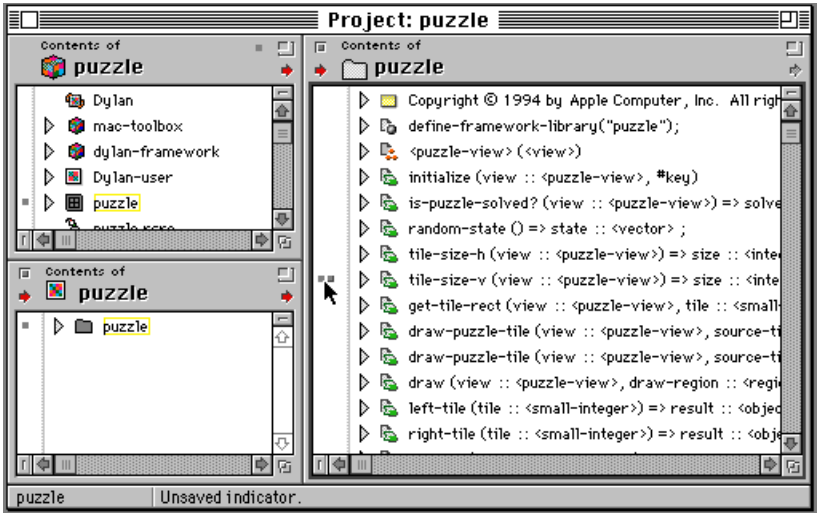
The Location field allows you to select where the status indicators will appear, in the status indicator column (choose Body) or in a pane's header (choose Header). You can choose to have them appear in both places if you wish.



The following figure shows a typical project with the Uncompiled indicator chosen and pointed to in the status indicator column. Note that the cursor is pointing at the green (second) indicator, which is the Uncompiled indicator. This causes the name of the indicator to appear in the prompt area at the bottom of the browser.



The following figure shows a typical project with the Unsaved indicator chosen and pointed to in the status indicator column. Note that the cursor is pointing at the blue (first) indicator, which is the Unsaved indicator. It is identified as the Unsaved indicator in the prompt area at the bottom of the browser.



Style — Text

Changes the font style of the selected source text. This choice changes only the text in a source record and has no semantic impact, but it does flag the record for recompilation. You can use font style to help organize your sources. For example, it is common to use bold for the name of objects being defined.

The font styles available are Plain, Bold, Italic, Underline, Outline, Shadow, Condense, and Extend.

Key shortcuts:

- Plain—**Command-Shift-T**
- Bold— **Command-Shift-B**
- Italic—**Command-Shift-I**
- Underline—**Command-Shift-U**
- Outline—**Command-Shift-O**
- Shadow—**Command-Shift-S**
- Condense—**Command-Shift-Option-C**
- Extend—**Command-Shift-E**

See "Formatting commands" on page 70.

You can change the font styles for the Listener and Browsers (as well as the source font style) using Preferences.

Subprojects (Aspect) — Browse

Operates on the active project and its subprojects, listing their subprojects.

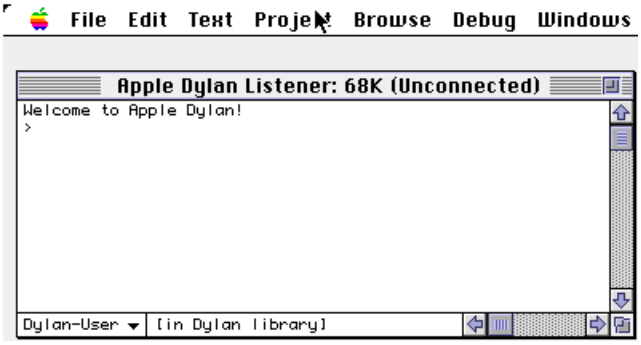
Note
Remember that a subproject is simply a project that has been included in another project.

Subprojects (Aspect) is commonly used to customize a browser pane displaying that aspect of project.

See Aspect and "Customizing browsers" on page 58 for more information.

Target Architecture — Project

Identifies the target platform, either 68K or PowerPC, for any projects built into applications or libraries. The default target is 68K. The target machine chosen remains set across all projects, whether they are currently open or not. If, for instance, a closed project had a 68K target when last opened and then is reopened after the target has been changed to PowerPC, the project will be retargeted to PowerPC. If a project is active when the target machine is changed, the application nub is quit and the name shown in the Listener changes to reflect the new target.



After you switch architectures, you must use the Update Project command to refresh the compiler results database and library model files. Otherwise, they will reflect the state they were in the last time the architecture was switched.

There is no provision in Apple Dylan for automatically using different source code for the two target machines.

Related commands: Set Project Type, Launch Application Nub, Quit Application Nub.

Tether to Application — Project

Tethers the development environment to a running, standalone application (which was originally created using the Create Application command from within Apple Dylan). The standalone application can then be debugged. The compiler results database for the original project must be available to tether to a standalone application. The application can either be on the same machine as

Apple Dylan or it may be running on some other system to which you have access.

The standalone application must be halted before you can use this command. You can cause it to halt by setting the event-check variable to be your own method for detecting a break request. For more information on event-check, see the book *Apple Dylan Extensions and Framework Reference*.

Here is an example of typical usage: if you get an error while running a standalone Dylan application, launch Apple Dylan and issue the Tether to Application command. You can then debug the application.

The notation "(Unconnected)" disappears from the Listener when the project's compiler results database file is tethered to the Application Nub.

See "Apple Dylan User Model" on page 95.

Related commands: Create Application, Launch Application Nub, Quit Application Nub.

Text of (Aspect) Browse

Operates on warnings in the active project, listing inline the expanded long text of the selected warning.

Text of (Aspect) is commonly used to customize a browser pane displaying that aspect of a warning. You can also select a single warning in a pane and display its expanded long text inline. See Aspect and "Customizing browsers" on page 58 for more information.

Uncompiled Source Folders (Aspect) Browse

Operates on modules, projects and subprojects in the active project, listing the source folders in the active project that contain uncompiled sources.

Uncompiled Source Folders (Aspect) is commonly used to customize a browser pane displaying that aspect of a module or project. See Aspect and "Customizing browsers" on page 58 for more information.

Uncompiled Source Records (Aspect) Browse

Operates on containers in the active project containing source records, listing the uncompiled source records they contain. Source record objects are methods, classes, variables, constants, macros, generic functions, comments, and top-level forms.

Uncompiled Source Records (Aspect) is commonly used to customize a browser pane displaying that aspect of a container. See Aspect and "Customizing browsers" on page 58 for more information.

Uncompiled Modules of (Aspect) Browse

Operates on a project or subproject in the active project, listing the modules containing uncompiled sources.

Note
Remember a subproject is simply a project that has been included in another project.

Uncompiled Modules of (Aspect) is commonly used to customize a browser pane displaying that aspect of a project. See Aspect and "Customizing browsers" on page 58 for more information.

Undefined Variables Browse

Opens a browser that lists the undefined variables in the active project.

Undefined Variables (Aspect) Browse

Operates on a container in the active project, listing the undefined variables in that container.

Undefined Variables (Aspect) is commonly used to customize a browser pane displaying that aspect of a container. See Aspect and "Customizing browsers" on page 58 for more information.

Undo — Edit

Undoes your last edit.

Key shortcut: **Command-Z**

Undo, Clear, Redo Clear, and Undo Clear replace one another on the File menu, depending on which is appropriate.

See "Editing in Apple Dylan" on page 66.

▲ **WARNING**
Clear supports Undo and Redo only for text. If you have cleared an object, folder, or subproject, it cannot be undone or redone. If you are about to clear a subproject, you are warned. You cannot clear required subprojects.

Undo Clear — Edit

Undoes a Clear by reinstating the text or object you had deleted. You can use Redo Clear to delete the text again, after using Undo Clear. Undo, Clear, Redo Clear, and Undo Clear replace one another on the File menu, depending on which is appropriate.

See "Editing in Apple Dylan" on page 66.

▲ **WARNING**
Clear supports Undo and Redo only for text. If you have cleared an object, folder, or subproject, it cannot be undone or redone. If you are about to clear a subproject, you are warned. You cannot clear required subprojects.

Undo More — Edit

Undoes your last edit up to 20 edits. The undos are in reverse order.

See "Editing in Apple Dylan" on page 66.

Unsaved Modules of (Aspect) Browse

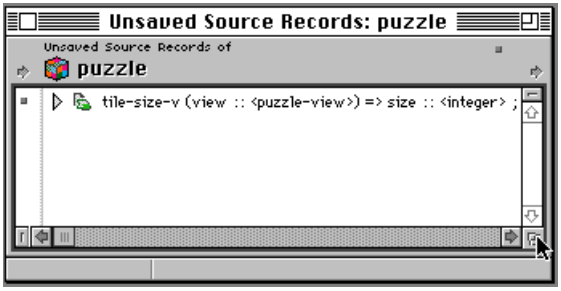
Operates on an uncompiled active project, listing the unsaved modules in it.

Unsaved Modules of (Aspect) is commonly used to customize a browser pane displaying that aspect of a project.

See Aspect and "Customizing browsers" on page 58 for more information.

Unsaved Source Records Browse

Opens a browser that lists all source folders that hold unsaved source records in the active project. The following figure shows the only unsaved source record in the puzzle project, tile-size-v.



Unsaved Source Records (Aspect) Browse

Operates on containers, listing source folders that hold unsaved source records in the active project.

Unsaved Source Records (Aspect) is commonly used to customize a browser pane displaying that aspect of a project.

See Aspect and "Customizing browsers" on page 58 for more information.

Update Project Project

Compiles any source records in the active project that have changed since the last update or recompilation and downloads them to the Application Nub, if it is tethered to the development environment. Any orphan definitions are

eliminated with this command. Orphan definitions are caused when source records are created, compiled, and then deleted.

Key shortcut: **Command-U**

Only the active project can be updated. Use the command Activate Project to make any open project the active project.

See the Project Interaction sheet under Preferences to choose whether to update a project automatically when it is opened.

Related commands: Recompile, Revert.

Variable Definitions of (Aspect) Browse

Operates on source records or definition entities in the active project, listing all variable definitions they include.

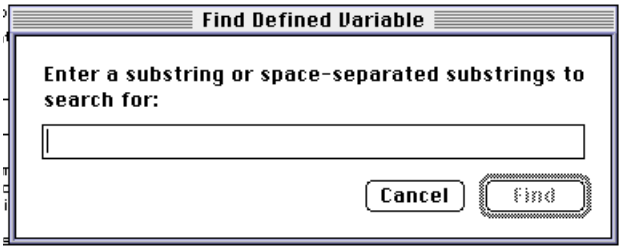
Variable Definitions of (Aspect) is commonly used to customize a browser pane displaying those aspects of an object.

See Aspect and “Customizing browsers” on page 58 for more information.

Variable Search Browse

Finds all variables in the active project that contain the entered string.

▲ **WARNING**
This is not a text search, but a search of the compiler results database. Uncompiled variables and text in comments will not be found.



Warning Source Record of (Aspect) Browse

Operates on warnings in the active project, listing the sources for the selected warning.

Warning Source Record of (Aspect) is commonly used to customize a browser pane displaying those aspects of an object.

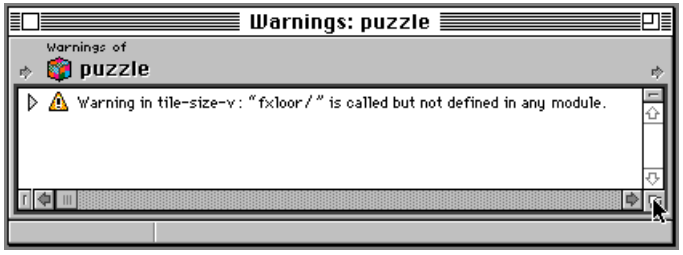
See Aspect and "Customizing browsers" on page 58 for more information.

The Warnings command opens a single-pane browser displaying all warnings for the active project. Warning Source Record of shows source records for selected warnings in this browser.

Warnings Browse

Opens the Warnings browser listing any objects in the active project that have generated warnings. The warnings are included. You can have more than one Warnings browser open at a time.

The following figure shows a typical Warnings browser.



Warnings for Downloaded Code of (Aspect) Browse

Operates on any object in the active project, listing warnings for any downloaded code. The types of warnings listed are compiler, linker, runtime errors, and runtime notices.

The Warnings command opens a single-pane browser displaying all warnings for the active project. Warnings for Downloaded Code of (Aspect) shows source records for selected warnings in this browser.

Warnings (for Selection) Browse

Displays a browser that lists all warnings for the selected object.

The Warnings command opens a single-pane browser displaying all warnings for the active project.

Warnings of (Aspect) Browse

Operates on source records and containers in the active project, listing all associated warnings. The types of warnings listed are compiler, linker, runtime errors, and runtime notices.

The Warnings command opens a single-pane browser displaying all warnings for the active project. Warnings of (Aspect) shows source records for selected warnings in this browser.

Glossary

.π – the suffix of an Apple Dylan project document.

.dl – the suffix of a Dylan library file.

.dylan – the suffix of a Dylan text file.

.fasl – compiled Lisp file used by the Apple Dylan development environment.

.lisp – Lisp file used by the Apple Dylan development environment.

.rsrc – the suffix of a Macintosh **resource file**.

active project – the open project that can be compiled and downloaded to the application nub. There can be only one active project at a time. Only the active project has its **compiler results database** loaded in the development environment. All relationships and aspects of the active project are available through the browsers. Inactive projects offer limited browsing. The active project is sometimes called the **root project** because all its **subprojects** are also active.

Apple Dylan – the Dylan development environment created by Apple, including the **framework** and **Creole**.

Apple Dylan Listener – the window that allows you to interact with the your application as it runs. See **connected**, **tethered**, **application nub**, **runtime**.

application – Internally, applications and libraries built by Apple Dylan are identical, except that an application includes a startup function. An application can be started and achieves some end result.

application nub – a tiny application that serves as the core of your Dylan program. The nub is essentially an empty Dylan application. See **runtime**.

aspect – a view of a particular relationship between objects. Aspects provide links between objects that describe relationships in a system. Source code is an obvious aspect of an object. Another aspect of an object is all the compiler linker, and runtime errors or warnings associated with it. All objects referenced by the source code of an object are aspects of that object, as are all objects containing a reference to that object. You can view all defined aspects of an object using a browser. The Aspects menu appears on the Browser menu.

auto-indentation – services that automatically indent source code for readability. Controlled from the Editing Sheet of the Preferences command.

automatic memory management – a system that reclaims objects from memory when they can no longer be referenced. This also known as garbage collection. When automatic memory management is running a “GC” icon replaces the cursor.

basis – the object that forms the input for a browser pane. The basis of a pane is displayed in the pane header.

browser – a paned window where you create and examine Apple Dylan objects. Typically, browsers comprise a set of linked panes displaying successively finer detail about a project or object. For example, the default project browser has three panes, the first showing the libraries and modules in a project, linked to a pane showing the **source folders** in one of those modules, linked to a pane showing the objects in that folder. If you change the selection in any of the first two panes, the succeeding panes change accordingly. See **hot link**.

call tree – the references to an object and references from an object are **aspects** that can be combined to create a graphic display of that object's calling relationships called a call tree.

cold link – See **hot link**.

compiler results database –the database that holds object code, debugging information, and other information generated by the compilation.

connected – in the Listener, short for connected to the **runtime,** which means the Application Nub is connected to the development environment. See **tethered**.

container – any object that contains other Apple Dylan objects, such as **projects**, **subprojects**, **modules**, and **source folders**.

Creole – feature of Apple Dylan providing cross-language support, used primarily for accessing the Macintosh Toolbox and other existing C libraries.

definition entity – the compiled representation of an object. In most uses of Apple Dylan you are looking at the object as compiled. However, each object has at least one definition entity and may have more, as in the case of a class, which has a definition entity for each slot. It is rare that you will need to work directly with a definition entity, but certain browsers may display them, using the bare icon for that kind of object. (See "Orphan definitions in the runtime" on page 111, "Apple Dylan Listener" on page 113, and "Using icons in Apple Dylan" on page 88.

development environment – the set of browsers, commands, and tools that support programming activities, also called **Apple Dylan**. The development environment can link to any application or library built with Apple Dylan.

download – to move compiled code from the development environment to the application nub. Performed automatically by the development environment as part of updating, compiling, or recompiling.

Dylan text file – a file that contains a text version of Dylan source code with the .dylan suffix. Used for representing Dylan source code outside the Apple Dylan development environment.

Dylan-user module – A source module included in all projects. In the simplest projects, it contains all the code for the project. In more complex projects, it contains the module and library definitions used to structure the rest of the project.

Emacs - an editor available on non-Macintosh platforms. Emacs-style editing has been partially implemented in Apple Dylan using Fred. See also **Fred**.

export – 1. The Export command on the File menu creates a **Dylan text file**.

2. In Dylan programming, you must write code to export modules from libraries and export variables from modules to make them importable for use by your project or by other modules, respectively.

expression - Anything that returns a value is considered an expression in Apple Dylan.

framework – an object-oriented class library that implements a common set of features found in Macintosh applications, such as the Clipboard, text editing, an event loop and event handling, scrolling and tracking, error reporting and event handling, resource management and streams. Just as the application nub is an empty Dylan application, the framework can be thought of as a generic, content-free Macintosh application.

Fred - an editor used to emulate Emacs-style editing. Fred stands for "Fred Resembles Emacs Deliberately". See also **kill ring**.

garbage collection – see **automatic memory management**.

hot link – Panes so linked that when you select an object in the first, the contents of the object are displayed in the second pane are said to be hot-linked. When you split a pane by using a splitter control, the two panes are automatically hot-linked.

inactive project – a project that is open, but cannot be compiled or downloaded because its database is not loaded in the development environment. You can open an inactive project to browse its text or copy code from it, but you cannot use most other browser features. See **active project**.

inbox – the arrow at the left of a pane's header that represents the input to the pane, that is, what is displayed there. See **outbox**.

incremental compilation – Apple Dylan supports changing and compiling any complete expression. You need not compile all possible code to make a change. You can compile **projects**, **source modules**, **source folders**, **source records**, and individual **expressions**. (You can even compile and execute code that has been commented out.)

information line – the region at the bottom of a browser window that provides ephemeral information depending on what you have selected. Also called the prompt area. If you have selected a function, its arguments are displayed. If you are dragging an object, you are told whether it can be dropped or not. In general, the information line gives feedback on current activities in the browser.

inline – In the development environment, certain commands or mouse gestures display their results in the same pane, or inline, as opposed to a separate pane.

inspector window – a window that displays information about objects in the **runtime**.

kill ring - the means of copying text as you edit in Fred. This is analogous to the Clipboard, but sometimes conflicts with it. See also **Fred**.

launch application nub – start the nub running and tether the development environment to the application nub. See **connected**, **tethered**, **runtime**.

library – a set of linked modules built in Apple Dylan with the Create Library command.

Listener – See **Apple Dylan Listener**.

load order – the order of the source records within a **source folder** and the order of source folders within a module and the order of modules within a project indicates the load order.

Two-pass compilation eliminates load-order restrictions of the type solved by forward declarations in other languages, but true load-order dependencies must still be addressed.

meter expression – to track the use of an expression.

module – a namespace for variable names. Items can be exported and imported between modules. Only specifically exported variables are visible outside of their module. Modules are contained in **projects**. Modules contain **source folders**.

nub – see **application nub**.

object-oriented dynamic language – Programming language supporting self-identifying objects, with automatic memory management, dynamic linking, and incremental development.

OODL – object oriented dynamic language

open project – a project with a project browser available. See **active project**.

orphan definition – a definition that exists in the runtime but has no apparent source. This can be caused when a source record that defines an entity is compiled and then subsequently renamed or deleted. Another cause is when the definition is defined, compiled and downloaded in the Apple Dylan Listener.

outbox – the arrow at the right of a pane's header that represents the output from the pane. Whatever is selected in the pane drives the context of the next links.

pane – Each **browser** consists of one or more panes. Each pane has a **basis** (the object displayed in the header region) and a default **aspect** (the kind of information displayed about the object).

project – an application or library under development. The project document has a suffix of π.

project file – the file in the Finder that contains pointers into all the documents that make up a project.

project browser – browser whose root pane has a project as its basis. The default project browser has three panes.

prompt area - See **information line**.

resource file – a standard Macintosh resource file with the suffix .rsrc. Use ResEdit or another resource editor to change the contents of a resource file.

root project – The **active project** is sometimes called the root project because it makes all its subprojects active as well.

runtime – everything that has been downloaded to the **application nub** or that has been built into an application. When the development environment is **tethered** to the application nub, or to a running application, it is said to be tethered to the runtime.

select, selection– Using the mouse, you can select **modules**, **source folders**, **source records**, or text within source records. Most actions in the Apple Dylan development environment depend on what you have selected. You can create a browser by selecting an object and double-clicking it. You can examine the union of aspects of several objects by selecting them all. You can compile individual source records or portions of source text through selection. Whatever is selected is highlighted.

source database – all sources for a project are included in this database. This database stores all the source code seen through the browsers. See **compiler results database**.

source folder – a container within a **module** that contains individual **source records**. Source folders are used primarily for organizing your project, but the order in which they are listed determines **load order**. There is no Dylan language construct corresponding to the source folder.

source module – See **module**.

source record – the container that holds a single piece of source code, usually representing a single object. Source records are contained in **source folders**.

splitter control– splits a pane. The horizontal splitter control is a short, horizontal bar just above the scroll bar on the right of a pane. The vertical splitter control is a short, vertical bar just to the left of the scroll bar at the bottom on a pane.

status indicator – a small colored square that appears in a browser pane or its header to indicate the status of an item or object.

status indicator column – the gray bar at the left of a browser pane where the status indicators appear.

subproject – a project that has been included in another project.

tethered – short for tethered to the **runtime**, or application nub. To be tethered means that the application nub is running, and Apple Dylan is connected to it.

unconnected – in the Listener, short for unconnected from the **runtime**. See **runtime**.

update – to automatically compile all uncompiled code and **download** it to the application nub. The action of the Update Project command.

user-interface builder – the portion of the development environment that helps you create a user interface for an application.

Index

A

Abort command 176
About Apple Dylan command 176
Activate Project command 176
Active project 273
Adding
  framework 131
  new user interface 163
  resource files 131
  subprojects 131
  user-interface builder 162
Add to Project command 177
All Methods of (Aspect) command 177
All Slots of (Aspect) command 177
All Subclasses of (Aspect) command 178
All Superclasses of (Aspect) command 178
Apple Dylan 273
  bailing out 117
  running 8
  running applications in 144
  user model 95
Apple Dylan Listener 113, 116, 273, 276
Apple Dylan Listener command 178
Apple menu
  About Apple Dylan command 176
  Hide Interface Builder command 214
  Show Interface Builder command 257
Application Nub 107, 273, 276
  launching 276
  tethering Apple Dylan to 6
Application Nub Info command 181
Applications 1, 2, 273
  building 164
  building standalone 118, 164
  running 15, 144
  setting characteristics 104, 126
  tethering to 145
Argument List (Copy Special) command 182
Argument List (Insert Special) command 182
Aspect 273
Aspect command 183
Aspects
  changing 46
  creating browsers for 5
  showing 42
Auto-indentation 273
Automatic memory management 273

B

Bailing out 117
Basis 2, 34, 274
Break command 189
Browse menu
  All Methods of (Aspect) command 177
  All Slots of (Aspect) command 177
  All Subclasses of (Aspect) command 178
  All Superclasses of (Aspect) command 178
  Aspect command 183
  Call Grapher command 190
  Classes of (Aspect) command 197
  Class Grapher command 193
  Contents of (Aspect) command 202
  Direct Methods of (Aspect) command 206
  Direct Methods of command 206
  Direct Slots of (Aspect) command 206
  Direct Slots of command 206
  Direct Subclass of (Aspect) command 207
  Direct Superclasses of (Aspect) command 207
  Duplicate Definitions command 207
  Duplicate Definitions of (Aspect)
    command 208
  Function Family (Aspect) command 213

Function Family command 212
Functions of (Aspect) command 213
Grapher Pane command 213
Info for Selected Class command 215
List of Browsers command 228
Look Up in Online Reference command 230
Modules of (Aspect) command 234
New Browser command 234
References From (Aspect) command 249
References From command 249
References To (Aspect) command 250
References To command 249
Resource Files of (Aspect) command 251
Save Browser command 253
Show Home command 256
Source Code of (Aspect) command 258
Source Folders (Aspect) command 259
Source Records (Aspect) command 259
Source Records with Warnings (Aspect) command 259
Status Indicators command 260
Subprojects (Aspect) command 263
Text of (Aspect) command 265
Uncompiled Modules of (Aspect) command 266
Uncompiled Source Folders (Aspect) command 265
Uncompiled Source Records (Aspect) command 266
Undefined Variables (Aspect) command 266
Undefined Variables command 266
Unsaved Modules of (Aspect) command 268
Unsaved Source Records (Aspect) command 268
Unsaved Source Records command 268
Variable Definitions of (Aspect) command 269
Variable Search command 269
Warnings (for Selection) command 271
Warnings command 270
Warnings for Downloaded Code of (Aspect) command 270
Warnings of (Aspect) command 271
Warning Source Record of (Aspect) command 270

Browsers 2, 274
  built-in 24
  creating for a specific aspect of an object 5
  customizing 58
  default project 2
  inbox 275
  information line 275
  linking 34
  project 2, 276
  saving configurations 61
  using 15
    Info for Selected Class 53
    References To 49
Building
  applications 164
  libraries 164, 166
  standalone applications 164

C

Call Grapher command 190
Call Recording command 191
Call tree 274
C code
  including 132
Checking
  code status 136
Classes of (Aspect) command 197
Class Grapher command 193
Class Template (Copy Special) command 195
Class Template (Insert Special) command 196
Clear command 69, 197
Close command 198
Code
  C
    including 132
  checking status 136
  compiling
    excluding from 143
    from Listener 141
    including in 143
  editing 76

importing 171
retrieving by importing 171
sharing 168
by exporting 168
uncompiled
compiling 140
Cold link 274
Collapse command 198
Color command 199
Compact Project command 200
Compilation
incremental 275
Compile and Download Region command 200
Compile and Download Selection command 200
Compile Expression command 201
Compile Region command 201
Compiler results database 1, 2, 274
Compile Selection command 202
Compiling
code
excluding 143
including 143
from Listener 141
projects 133
selections 139
uncompiled code 140
Container 274
Containers 5
Contents of (Aspect) command 202
Continue command 203
Copy command 67, 203
Copy Special command 68, 203
Copy Title Text command 204
Create Application command 204
Create Library command 205
Creole 274
Customizing
browsers 58
development environment 83
Cut command 67, 205

D

Debugging
projects 146
Debug menu
Abort command 176
Break command 189
Call Recording command 191
Continue command 203
Expand Macro command 210
Inspect Heaps command 216
Inspect Listener Result command 218
Inspect Module Variables command 218
Inspect Selection command 221
Meter Expression command 231
Pattern Match Macro command 242
Pattern Match Macro Including Builtin command 242
Reset Stack command 250
Defaults
setting
editing 86
Listener interaction 87
Definition entity 274
Development environment 274
customizing 83
introduction 1, 2
running 8
setting preferences 84
Direct Methods of (Aspect) command 206
Direct Methods of command 206
Direct Slots of (Aspect) command 206
Direct Slots of command 206
Direct Subclass of (Aspect) command 207
Direct Superclasses of (Aspect) command 207
Disclosure triangles 3
Duplicate Definitions command 207
Duplicate Definitions of (Aspect) command 208
Dylan text files 274
exporting 70
importing 70
Dylan-user module 274

E

Editing 66
  code 76
  commands
    Emacs-style 66, 72, 76
    Macintosh-style 71, 72
  setting defaults 86
  tools 66
Edit menu
  Argument List (Copy Special) command 182
  Argument List (Insert Special) command 182
  Class Template (Copy Special) command 195
  Class Template (Insert Special) command 196
  Clear command 69, 197
  Collapse command 198
  Copy command 67, 203
  Copy Special command 68, 203
  Copy Title Text command 204
  Cut command 67, 205
  Expand command 209
  Insert Special command 68, 216
  Method Template (Copy Special)
      command 232
  Method Template (Insert Special)
      command 233
  Paste command 67, 241
  Preferences command 243
  Redo Clear command 248
  Redo command 69
  Select All command 253
  Undo Clear command 267
  Undo command 69, 267
  Undo More command 69, 267
Emacs 66, 275
  editing commands 72
Exclude Source Records command 208
Expand command 209
Expand Macro command 210
Export command 211
Exporting 275
  code
    sharing by 168
  Dylan text files 70
Expression 275
Expressions
  executing in Listener 5
  metering 156

F

Fat applications 103
File menu
  Add to Project command 177
  Close command 198
  Export command 211
  Import command 214
  New Module command 120, 237
  New Project command 238
  New Source Folder command 239
  New Source Record command 239
  New Text Window command 240
  Open command 241
  Page Setup command 241
  Print command 247
  Quit command 247
  Revert command 251
  Save All command 252
  Save command 252
Find Again command 212
Find command 69
Find/Replace command 211
Find Selection command 212
Font command 212
Formatting commands
  Color 70
  Font 70
  Size 70
  Style 70
Framework 275
  adding 131
Fred 72, 275
  kill ring 276
Function Family (Aspect) command 213
Function Family command 212
Functions

monitoring 159
Functions of (Aspect) command 213

G

Garbage collection 275
Grapher Pane command 213

H

Heaps
inspecting 153
Hide Interface Builder command 214
Hot link 275

I

Icons
using 88, 93
Import command 214
Importing
code
retrieving by 171
Dylan text files 70
Inactive project 275
Inbox 275
Include Source Records command 214
Incremental compilation 275
Info for Selected Class browser 53
Info for Selected Class command 215
Information line 4, 275
Inline 275
Insert Special command 68, 216
Inspect Heaps command 216
Inspecting
heaps 153
Listener results 151
modules 154
stack 148
Inspect Listener Result command 218
Inspect Module Variables command 218
Inspector windows 116, 117, 275
Inspect Selection command 221
Inspect Stack command 221

K

Key command shortcuts 173, 175
Kill ring 276

L

Launch Application Nub command 227
Launching
runtime 135
Libraries 276
building 164, 166
setting characteristics 128
version numbers 106
Linking
browsers 34
panes 3, 34
Links 34
cold 274
hot 275
inbox 34
Listener 5, 273, 276
compiling code from 141
inspecting results 151
setting interaction defaults 87
Listener window 113, 116
List of Browsers command 228
Load order 120, 126, 237, 276
Load UI Builder command 230
Look Up in Online Reference command 230

M

Meter Expression command 231
Meter expressions 276
Metering
  expressions 156
Method Template (Copy Special) command 232
Method Template (Insert Special) command 233
Module
  source 277
Modules 276
  inspecting 154
Modules of (Aspect) command 234
Monitoring
  individual functions 159

N

New Browser command 234
New Module command 120, 237
New Project command 238
New Source Folder command 239
New Source Record command 239
New Text Window command 240

O

Object-oriented dynamic language 276
Objects
  saving in panes 130
OODL 276
Open command 241
Open project 276
Orphan definitions 111, 276
Outbox 276
outbox 34

P

Page Setup command 241
Panes 2, 276
  basis 2, 34
  disclosure triangles 3
  information line 4
  Linking 34
  linking 3
  outbox 276
  root 2
  saving objects in 130
  splitter control 277
Paste command 67, 241
Pattern Match Macro command 242
Pattern Match Macro Including Builtin
    command 242
Preferences command 84, 243
Print command 247
Project browser 276
Project files 276
Project menu
  Activate Project command 176
  Application Nub Info command 181
  Compact Project command 200
  Compile and Download Region command 200
  Compile and Download Selection
    command 200
  Compile Expression command 201
  Compile Region command 201
  Compile Selection command 202
  Create Application command 204
  Create Library command 205
  Exclude Source Records command 208
  Include Source Records command 214
  Inspect Stack command 221
  Launch Application Nub command 227
  Load UI Builder command 230
  Quit Application command 247
  Quit Application Nub command 247
  Recompile command 248
  Run command 251
  Set Project Type command 253
  Target architecture command 264

Tether to Application command 264
Update Project command 268
Projects 95, 276
active 102, 273
applications
setting characteristics 126
browsing 16
compiling 133
creating new 120
debugging 146
files 101
inactive 102, 275
libraries
setting characteristics 128
open 276
opening 11, 101
root 102, 277
saving 130
scope 97, 100
setting characteristics 104
applications 104
libraries 104
sharing 168
starting 119
synchronization 109
orphan definitions 111
restoring 113
Prompt area 276

Q

Quit Application command 247
Quit Application Nub command 247
Quit command 247

R

Recompile command 248
Redo Clear command 248
Redo command 69
References From (Aspect) command 249
References From command 249
References To (Aspect) command 250
References To browser 49
References To command 249
Replace and Find command 250
Replace command 69
Reset Stack command 250
Resource files 276
adding 131
Resource Files of (Aspect) command 251
Revert command 251
Root project 277
Run command 251
Runtime 277
launching 135
untethering from 136

S

Save All command 252
Save Browser command 253
Save command 252
Saving
objects in panes 130
projects 130
Select All command 253
Selection 277
Selections
compiling 139
Set Project Type command 253
Sharing
code 168
by exporting 168
projects 168
user interface 164
Show Home command 256
Show Interface Builder command 257
Size command 258
Source Code of (Aspect) command 258
Source database 277
Source folder 277
Source Folders (Aspect) command 259

Source module 277
Source records 1, 2, 277
Source Records (Aspect) command 259
Source Records with Warnings (Aspect) command 259
Splitter control 277
Stack
  Inspecting 148
Standalone applications
  building 118
Status
  code
    checking 136
Status indicator column 277
Status indicators 277
  and synchronization 111
Status Indicators command 260
Style command 263
Subprojects 277
  adding 131
Subprojects (Aspect) command 263
Synchronization 109
  restoring 113

T

Target Architecture 103
Target Architecture command 264
Tethering 6, 277
  to running applications 145
Tether to Application command 264
Text menu
  Color command 199
  Find Again command 212
  Find command 69
  Find/Replace command 211
  Find Selection command 212
  Font command 212
  Replace and Find command 250
  Replace command 69
  Size command 258
  Style command 263
Text of (Aspect) command 265
Two-machine development 9, 120, 244

U

Uncompiled Modules of (Aspect) command 266
Uncompiled Source Folders (Aspect) command 265
Uncompiled Source Records (Aspect) command 266
Undefined Variables (Aspect) command 266
Undefined Variables command 266
Undo Clear command 267
Undo command 69, 267
Undo More command 69, 267
Unsaved Modules of (Aspect) command 268
Unsaved Source Records (Aspect) command 268
Unsaved Source Records command 268
Untethering
  from runtime 136
Update Project command 268
Updating 277
User interface
  adding new 163
  creating 161
  sharing 164
User-interface builder 277
  adding 162
User model 95

V

Variable Definitions of (Aspect) command 269
Variable Search command 269

W

Warnings (for Selection) command 271
Warnings command 270

Warnings for Downloaded Code of (Aspect)
command 270
Warnings of (Aspect) command 271
Warning Source Record of (Aspect)
command 270
Windows menu
Apple Dylan Listener command 178

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Proof pages were created on an Apple LaserWriter Pro printer. Final pages were created on a Docutek. Line art was created using Adobe Illustrator™ and Adobe Photoshop™. PostScript™, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.

PRODUCTION MANAGER
Trish Eastman

LEAD WRITER
Linda Kyrnitszke

WRITERS
Sarah Lee Bihlmayer, Tom Parmenter, Daphne Steck

ILLUSTRATOR
Sandee Karr

PRODUCTION EDITORS
Lorraine Findlay, Alexandra Solinski

SPECIAL THANKS TO
Jeremy Jones, Paige Parsons, Andrew Shalit