Apple Dylan Quickstart

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

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manual or in the media on which a software product is distributed, APDA will replace the media or manual at no charge to you provided you return the item to be replaced with proof of purchase to APDA.

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

CHAPTER 1

Introduction

Contents

About Apple Dylan

Apple Dylan is a development environment for building software for Macintosh computers. It combines an implementation of the Dylan programming language with a powerful browser-based development environment, and extensions for linking to the Macintosh toolbox and C object code. Apple Dylan also includes an application framework and user-interface builder designed to take advantage of the Dylan language and the Macintosh.

Dylan is an object-oriented dynamic language. A high-level programming model, thorough object orientation, and garbage collection work together to simplify program structure. In Dylan, you can think about your programs more clearly and write them more quickly. You can produce greater functionality with fewer lines of code. This adds up to fewer bugs, faster delivery, and lower maintenance costs.

The Apple Dylan development environment extends the power of Dylan with a flexible project model and browsing system. It provides a Finder-like user interface that allows you to use direct manipulation techniques to explore your program. The browsers let you view your code from a number of perspectives. For example, a project can be viewed as source code, as a network of classes, as a network of calling relationships. The browsers in Apple Dylan are customizable, so you can tailor your environment to your preferences. All development in Apple Dylan is incremental and interactive.

The Apple Dylan application framework builds on previous application framework designs, but leverages the unique capabilities of Dylan, yielding a framework that is easy to understand, use, and extend.

Apple Dylan's cross-language linking and calling facilities support automatic inclusion of C header files in Dylan programs. This lets programmers take advantage of the Macintosh toolbox as well as the great store of static language libraries that currently exist, all from the comfort of their Dylan programming environment.

About the Technology Release

This release allows programmers to start developing Apple Dylan applications while evaluating Apple Dylan technology.

The release does have a number of limitations:

- While applications and libraries written in Dylan will run native on the PowerPC (as well as on the 68K), the development environment itself is not PowerPC native.
- The software has not been fully debugged.
- There are many missing features in the development environment, framework, and user interface builder.
- The documentation is preliminary.

Even with these limitations we feel that Apple Dylan is an exciting tool for exploring new approaches to programming.

Reporting Bugs

When reporting bugs, it is helpful to include information about your Apple Dylan and Macintosh configuration. The easiest way to do this to select the "About Apple Dylan…" command from the Apple menu, and then click on the "Report Bug…" button in the dialog box which is shown. Clicking on this button will copy some text to the clipboard. You can then paste this text into an e-mail message as part of your bug report.

The e-mail address for bug reports is given in the "About Apple Dylan…" dialog box.

Please try to send bug reports with a meaningful subject field, to help us track your report better. For example, "type error while browsing" is more useful than "A Dylan bug".

Additional Dylan Information

There are a variety of sources of information about Dylan on the Internet and other information services.

- On the Internet, the Apple Dylan World-Wide Web site is http://www.cambridge.apple.com
- The Apple Dylan ftp site is cambridge.apple.com:/pub/dylan/
- The Dylan WWW page at Carnegie Mellon University is http://legend.gwydion.cs.cmu.edu:8001/dylan/
- The Dylan newsgroup is comp.lang.dylan
- The info-dylan and info-dylan-digest mailing lists contains the same messages as the comp.lang.dylan newsgroup. To subscribe to info-dylan or info-dylan-digest send mail to majordomo@cambridge.apple.com. The body of the message should be "subscribe <list-name>", where <list-name> is the name of the mailing list you want to subscribe to. To unsubscribe to one of the mailing lists, send majordomo a message with the body "unsubscribe <list-name>." If you would like to subscribe or unsubscribe an address which is different from the return address of the message, include the address after the <list-name>. For complete majordomo instructions, send a message with the body "help". Please do not send administrative requests to the mailing lists! If you have trouble with info-dylan, send mail to sysadmin@cambridge.apple.com.

CHAPTER 1

Introduction

CHAPTER 2

Installation

Contents

This chapter includes information about configuration requirements, compatibility, and instructions for installing the Apple Dylan software.

Configuration Requirements

Apple Dylan can be used in either a one-machine configuration, or a two-machine configuration. In the one-machine configuration, the Apple Dylan development environment and the program you are developing both run on the same machine. In the two-machine configuration, they run on different machines which are connected by an AppleTalk network.

We recommend running Apple Dylan on a 68040 Macintosh with at least 20 megabytes of *physical* memory. You can use virtual memory, but if you have less than 20 megabytes of physical memory, performance may be poor.

The Apple Dylan development environment requires a memory partition of at least 14.5 MB, with 18 MB suggested. A low memory version of Apple Dylan, "Apple Dylan SI" requires a memory partition of at least 10 MB, with 14 MB suggested. This decrease in memory requirements comes at some cost in performance. See "About Apple Dylan SI" below.

The actual amount of memory used by Apple Dylan varies with the operations being performed. For example, it takes significantly more memory to compile a project or create a library than to use a library of the same size.

The amount of memory required for the application you are developing depends on the characteristics of the application. It's probably a good idea to start with around 2 MB, and work down or up from there. If you are using the interface builder, you should start at about 4 MB and work up or down from there.

The user interface for the development environment is optimized for a 16-inch or larger 8-bit color monitor. However, the system is usable on any type of monitor including smaller monochrome monitors.

The Apple Dylan development environment runs best on a 68030 or 68040 based Macintosh. To run the development environment on a Power Macintosh, you should turn off the modern memory manager.

Applications under development can be run on any Macintosh, and will run native on the PowerPC.

Apple Dylan has been tested on versions 7.1 and 7.5 of the Mac OS.

- **Note:** The Macintosh operating system does not always deal gracefully with situations where the system heap needs to expand, but cannot expand because all available memory has already been allocated to applications. For this reason, if your Macintosh has limited memory you should use the "Get Info" command on the Finder's "File" menu to set the preferred size of the application nub and of the Apple Dylan development environment. Set them so that when both applications are running there is still at least 50K unused, to leave room for system heap expansion. Use the Finder's "About This Macintosh" command to verify that enough unused memory has been left.

About Apple Dylan SI

Apple Dylan SI (Swapping Image) is a version of Apple Dylan that requires substantially less memory to run than Apple Dylan. It does this by keeping the code of the development environment on disk until it is called. This means that the first time you do anything in the swapping Apple Dylan it is substantially slower than the non-swapping version of Apple Dylan. The second time you do something, it is just as fast.

Swapped-in code stays in memory until the free space within Apple Dylan drops below 500K. When this happens, all code gets purged from memory, so subsequent operations will be slow again. If your free space remains consistently below 500K, the system stays very sluggish since code is constantly being purged and swapped in.

Apple Dylan SI supports browsing and editing in a partition as small as 10 MB. Simple projects that don't use the framework can be compiled in 12MB or less. If you're compiling projects that use the framework, you should allocate at least 14MB. Remember that Apple Dylan can always make good use of any additional memory you can afford to give it.

Contents of the Seed Release

The Apple Dylan technology release consists of several printed documents and a CD. The contents of the CD are:

1. **Installation Instructions**
   The latest and greatest installation instructions.
2. **Apple Dylan Documentation**
   A variety of documents, including online versions of the printed documentation in Acrobat and QuickView formats. This folder also includes the Acrobat Reader and QuickView application.
3. **Other Goodies**
   A folder containing several non-Apple Dylan implementations and some user-contributed Dylan code.
4. **Apple Dylan TR**
   The Apple Dylan development environment software and related files.

- **READ ME FIRST**
  A read-me file with late-breaking news and release notes.
- **Apple Dylan**
  The Apple Dylan development environment.
- **Apple Dylan SI**
  A version of the Apple Dylan development environment which requires less memory.
- **Extension for 030's**
  A folder containing a system extension that improves the performance of the Apple Dylan development environment when running on 68030-based Macintoshes. This extension is not necessary on other Macintoshes, and it is not needed to run applications produced by Apple Dylan.
- **Browsers**
  A folder of saved browser configurations used by Apple Dylan.
- **Sample Code**
  A folder containing a number of sample projects. *Note that there are additional samples in the framework folder.*

- **Apple Dylan Files**
  A number of libraries, files, and utilities used by Apple Dylan.
  - **Application Nub**
    A folder containing the Apple Dylan application nub, the Dylan library, and a utility program used to quit the application nub when other techniques fail.
  - **Framework 1.0d57**
    A folder containing the Apple Dylan application framework project, the Mac-Toolbox project (which imports some common toolbox definitions), the Dylan user interface building, and the framework sample applications.
  - **Cincludes**
    A folder containing Universal C Header files for the Macintosh Toolbox. These are the same header files that were included with ETO 17.
  - **Kernel**
    A folder containing source files and instructions for building your own custom application nubs. This is useful when you want to link C code to your Dylan application.
  - **Symbols** and **symbols.db**
    Files that contain the names of the symbols used internally by Apple Dylan. These files make bug reports more readable.

Installation Instructions

To install Apple Dylan, copy the "Apple Dylan TR" folder to your hard disk.

Inside the folder "Apple Dylan Files:Application Nub:" you will find eight files ending in the extension ".dl". Make aliases to these eight files and place the aliases in the Extensions folder of your startup disk.

If you will be running Apple Dylan on a 68030 Macintosh, you should copy the system extension from the "Extension for '030's" folder into your Extensions folder.

To enable access to the on-line reference documentation from Apple Dylan, copy the folder "Apple Dylan Documentation:QuickView:" to your hard disk.

The remainder of the contents of the CD can be accessed as needed.

Known Incompatibilities

Software written in Apple Dylan requires System 7 or later, but does not have the other incompatibilities.

The Apple Dylan development environment is not compatible with RAM Doubler, with the modern memory manager the Power Macintosh, and with versions of the Mac OS earlier than System 7.

The QuicKeys Special commands “Select rear window” and “Select second window” don't work in Apple Dylan. They bring the appropriate window to the front, but don't activate it.

We cannot guarantee compatibility with third-party hardware accelerators.

Apple Dylan has not been tested on A/UX or on Sun and HP workstations running MAE.

Please send bug reports describing any other problems you have.

Installation

CHAPTER 3

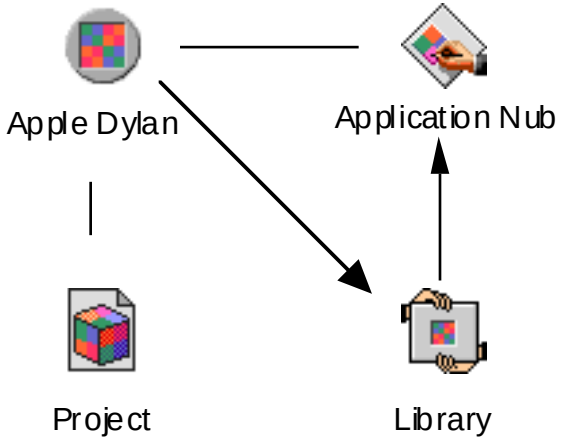
Tutorial

Contents

This chapter explains the fundamentals of Apple Dylan. It gives an overview of the system architecture, project structure, browsing tools, libraries, and debugging tools. It describes how to edit, compile, and debug software with Apple Dylan, how to create stand-alone applications, and how to use libraries.

Development Architecture Overview

Apple Dylan uses a cross-development architecture:



The development tools are all part of the **Apple Dylan** application. Software under development runs inside an **application nub**. The application nub is essentially an empty Dylan application. During the course of development, code is loaded into the application nub and executed. Apple Dylan communicates with the application nub via Apple Events. Apple Dylan and the application nub can be on the same machine, or on different machines. The application nub that is currently connected to Apple Dylan is sometimes referred to as the **runtime**.

Programs written in Apple Dylan are stored in **projects**. A project may be used to create an **application** or a **library**. Projects are described in greater detail below.

Project Structure

Projects consist of a project file, a source code database, compiler results databases, library models and files, pointers to subprojects, resources, header files and other files.

- The source code database holds the text of the Dylan program in the project.
- The compiler results databases hold the compiled form of the source code, and additional debugging information. There will be one compiler results database for each cpu architecture you target (68K and PPC).
- Library models and files contain linked object code. They are described in the next chapter.
- Subprojects are other Dylan projects which are used by the project. Subprojects may export modules which are imported by the project, or they may simply add behavior to generic functions and classes.
- Resources are Macintosh resources which are used by the program under development. These resources are loaded automatically by the application framework.
- Header files are C headers that you plan to import into your project using Creole. Note that all the Macintosh header files are included with Apple Dylan in the CIncludes folder. You only have to add header files to your project if they are not part of the Macintosh Toolbox.
- Other files may be added to a project for the convenience of the programmer, but are not treated specially by Apple Dylan.

The project file, source code database, compiler results databases and library models are all stored in a single project folder, and should not be modified by the programmer. The other project contents may also be stored in this folder, or they may be stored elsewhere. The project file records the locations. If the items are moved, Apple Dylan will prompt you to locate them when they need to be accessed.

Using a Project

The remainder of this chapter walks you through opening and browsing a project, running and modifying it, and creating an application from it.

This tutorial uses one machine development.

Opening the project

Launch the Apple Dylan application. When Apple Dylan is done starting up, a window titled “Apple Dylan Listener: 68K (Unconnected)” will appear.

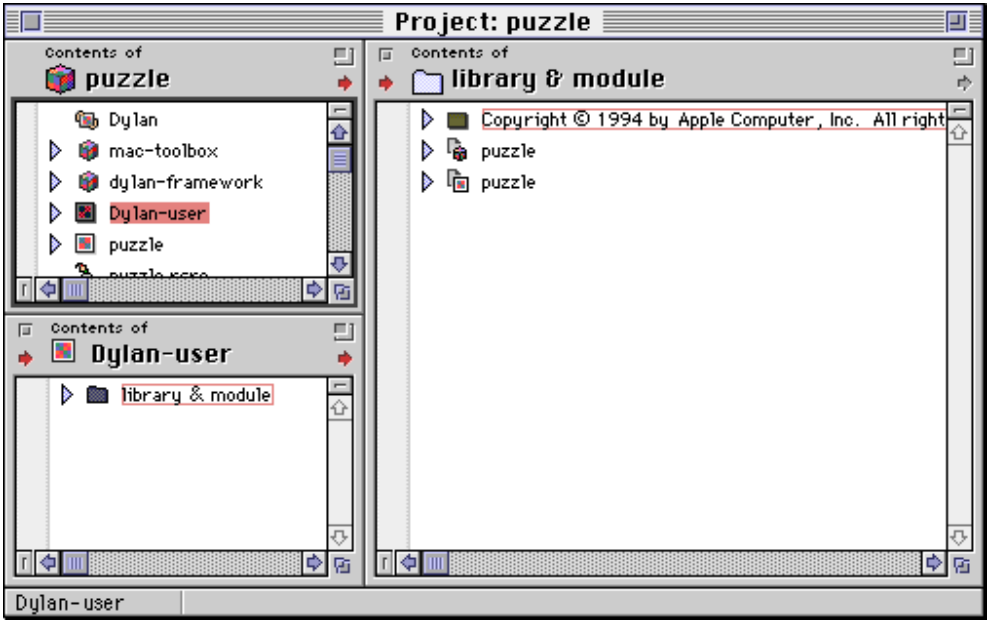
If you are developing on a PowerPC machine, choose “PPC” from the “Target Architecture” command on the “Project” menu. The title of the listener window will change to “Apple Dylan Listener: PPC (Unconnected)”

Open a project by selecting the “Open…” command from the “File” menu.

Open the “Puzzle” project. The path to this project is “Apple Dylan Files:Framework 1.0d57:samples:puzzle:puzzle.π”.

As the project is opened, a progress indicator is displayed. Opening the project locates and opens the project source database and loads information to allow the project to be browsed.

Beginning browsing



When the project is opened, it is displayed in a three-paned project browser. The top left pane shows the top level contents of the project, in this case three subprojects (“Dylan”, “mac-toolbox”, and “dylan-framework”), two modules (“Dylan-user” and “puzzle”), and a resource file (“puzzle.rsrc”). The top left pane is **linked** to the bottom left pane. This means that when an object is selected in the top left pane, its contents are displayed in the bottom left pane. Similarly, the bottom left pane is linked to the right pane.

- **Note:** You can modify or create your own browser configurations and save them. You can do this to create new browsers or to replace existing browsers. Creating browsers is described in “Using the Apple Dylan Development Environment.”

Once the progress indicator is gone, you can begin browsing your project.

Select the “puzzle” module in the top left pane.

You do this by clicking once on the name of the module or on the icon to the left of the name. You will see the contents of the puzzle module displayed in the bottom left pane. In this case, the contents consists of a single source folder,

called “puzzle”. This folder is automatically selected, displaying its contents, a series of source records, in the right hand pane. Each source record corresponds to one piece of source code or comment. To view the contents of one of the source records, click on the triangle to its left. The triangle will animate, revealing the contents of the source record (in this case, source code). Clicking on the triangle again will hide the contents. You can also double-click on a source record (or any other object) to see its contents in a separate window.

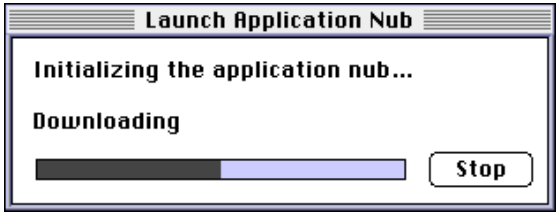
You can change the allocation of space among the panes in the window by using the grow boxes in the lower right corner of each pane. You can adjust the size of the entire window by using the grow-box of the lower-right pane or by using the window’s zoom box.

Running the project

Next we are going to run the code in the project, to see that the application it defines works.

Select the “Launch Application Nub” command from the “Project” menu.

This launches the application nub and instructs the nub to load the libraries included in the project (in this case, the Dylan kernel definitions and the framework). During this process, a progress indicator is displayed.



You can check to make sure the application nub has been launched correctly by looking on the application menu (the rightmost menu on the menubar).

In addition to Apple Dylan, Finder, and any other applications you have running, the application nub should be running. Switching to the application nub shows an application with no windows and no menus. You can return to Apple Dylan from the nub by clicking in one of Apple Dylan's windows.

Updating the project

Before working with your program interactively, you will usually want to **update** it. You do this by choosing the "Update" command from the "Project" menu. Updating recompiles any source code which has been edited since it was last compiled, and any dependent source records. If you are connected to the application nub, updating will also download any changes you've made, thereby synchronizing the application nub and your project sources.

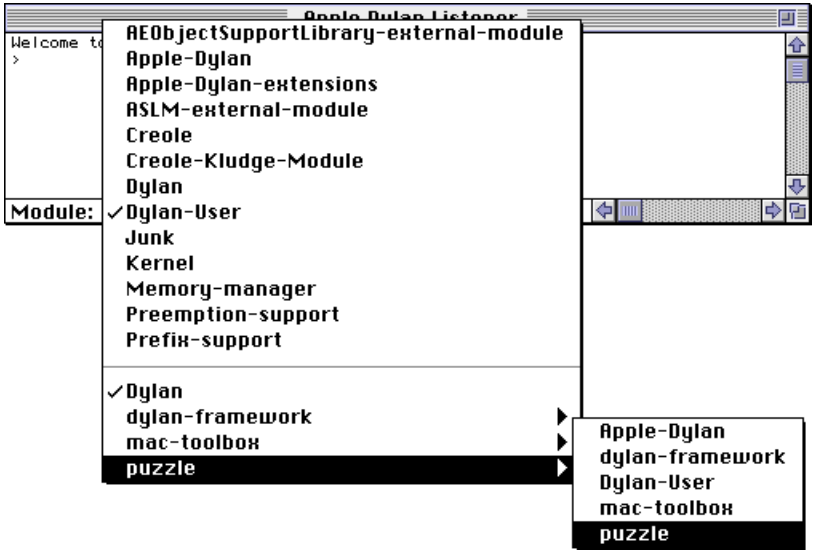
- **Note**: The sample project you're working with, "Puzzle" was updated and compiled before it was shipped to you. Choosing update on the Puzzle project should not cause any recompilation to occur. If doing an update results in a significant amount of recompilation, you may not have installed the software correctly, or you may have inadvertently modified a subproject such as the framework. In this case it is suggested that you recopy the project from the CD, and double check that your aliases are set correctly in the extensions folder.

Using the listener

After connecting, the name of the listener changes to “Apple Dylan Listener”; the word “disconnected” is removed. The listener now serves as a way to interact with the functions, classes, and bindings in the runtime. You can enter Dylan program fragments in the listener, and they will be compiled, downloaded to the runtime, and executed on the runtime. Any values returned by the execution of these program fragments will be printed in the listener.

The listener executes code as if it were a top level source record compiled in the module that the listener is set to. It does not have access to any local bindings, only module bindings.

Before typing program fragments into the listener, you need to make sure the listener is set to the proper module. Otherwise, the bindings you enter will not be compiled in the right scope. In this case, we will be testing code in the “puzzle” module which is in the “puzzle” library. To check that this module is selected, look at the pop-up menu in the lower-left corner of the listener.



Begin by typing some simple program fragments. At the end of each line, type return or enter.

```
> 10 + 10   // (spaces around + required)
20
> random-state()
#[7, 15, 8, 9, 6, 2, 13, 3, 5, 4, 0, 1, 10, 11, 14, 12]
> random()
16657
>
```

+ is a operator built into Dylan. `random-state` is a function defined by the puzzle project, and `random` is a toolbox trap that was imported into Dylan from a C header file.

You can look at the source code for `random-state` by selecting the name `random-state` in the listener and then selecting the "Show Home" command from the "Browse" menu. You may want to resize the window to see the source code more easily. Choosing "Show Home" on `random` will take you to the `define-interface` statement which imports the C header.

Next we'll run the application, to see what it does. The startup function for applications built using the Apple Dylan framework is called `start`. It first performs initializations as specified by the framework and the application, and then enters the main event loop. Because all the code for the puzzle application and the framework have been loaded into the application nub, we can start the application simply be calling the `start` function from the listener.

> **start()**

After initialization, the puzzle application switches itself to the front. Note that in the background, Apple Dylan is still showing a progress indicator. That's because it's waiting for the function `start` to return (which it won't do until we cause the puzzle application to return from its main event loop).

- **Note:** You can connect to the application nub, perform an update, and run the startup function in a single step by choosing the "Run" command from the "Project" menu.
- **Note:** You can partially reset a framework application by calling the `reset` function. This closes windows, deinstalls menus, and calls user defined reset methods. See the framework documentation for more details.

You can play with the puzzle to make sure it works, create new puzzles, etc. When you are done playing with the puzzle, choose the "Pause Application" command from the "Debug" menu of the puzzle application. This will cause

the main event loop of the puzzle application to return, and place the puzzle application in a state where it can respond to additional commands from Apple Dylan. The "Debug" menu is a special menu that framework installs when an application is under development.

Typing command-option-period with the puzzle as the front most application will also cause the main event loop to return. This is also a feature of the Apple Dylan framework, and is only available for projects built on the framework.

- **Note:** When a program hits an error or a break, the main event loop is similarly suspended, allowing the application under development to receive commands from Apple Dylan.

Exploring the program

Apple Dylan lets you execute Dylan one program fragment at a time in the context of your running program. This lets you explore your data structures, test out new functionality, and replace existing class and method definitions. Because you don't have to quit and restart your program, you can test corrections without having to go through the process of duplicating an error condition.

You can type temporary program fragments into the listener. To change a part of your program permanently, you edit its source code and recompile it from a browser window. If you are connected to an application nub, the code is downloaded to the runtime when it is compiled.

There are many ways to explore your program by invoking functions defined by Dylan, the framework, or your program itself. For example, to look at the front window, you could use the framework function `front-window`.

```
> front-window()
#<<window> id: 1>
```

To explore further from this window, select the "Inspect Listener Result" command from the "Debug" menu. This brings up an inspector window showing the contents of the window object. You can double click on the entries in the inspector window to explore further, and choose commands from the inspector pop-up menu for additional information (for example, to see all the objects which reference a given object). You can select a class or function in an

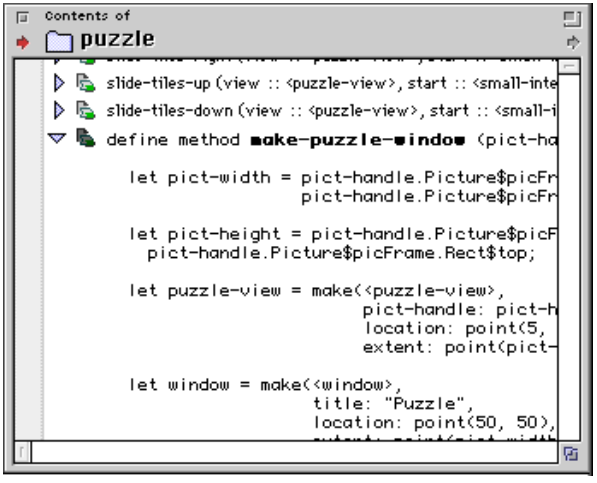
inspector window and choose the “Show Home” command from the “Browse” menu to see the source code of the class or function.

Modifying the program

Next we’ll make a change to the puzzle application and test the change.

Find the function `make-puzzle-window` in the right hand pane of the project window, and open it up to view its source code.

- **Note:** You can temporarily expand the right hand pane by clicking in the pane’s zoom box, which is in its upper right hand corner. This expands the pane to fill the entire window. Clicking the pane zoom box a second time restores the old pane configuration.



In the second call to `make`, change the keyword argument `title:` from `"Puzzle"` to `"Game"`.

Note that a blue box appears to the left of the title of the source record. This is a status indicator, indicating that the source record has been modified and not saved. The source folder “puzzle” is also marked as modified and unsaved, and the module “puzzle” is also so marked. There are other status indicators, for marking compiler warnings, uncompiled code, etc. Not all indicators are shown in all panes by default. You can control which status indicators are

shown in which pane. For more details, see the Apple Dylan User Interface chapter.

To save the source record, choose the “Save” command from the “File” menu. The status indicator will go away.

To recompile `make-puzzle-window` select the icon next to its title and type enter or choose the “Compile and Download Selection” command from the “Project” menu. When compilation and downloading are complete, Apple Dylan will print the following message in the listener:

```
defined make-puzzle-window (pict-handle :: <PicHandle>)
        => window :: <window> ;
```

The function has been recompiled, and the new version has been stored both on the runtime and in the compiler results database.

Now re-enter the application’s event loop by typing `start()` in the listener or selecting the “Run” command from the “project” menu. New windows that you create will have the title “Game” rather than “Puzzle”.

Disconnecting from the nub

Once again, type command-option-period with the puzzle as the front most application.

After execution has stopped and you have returned to Apple Dylan, select the “Quit Application Nub” command from the “Project” menu and choose “Quit” in the confirmation dialog.

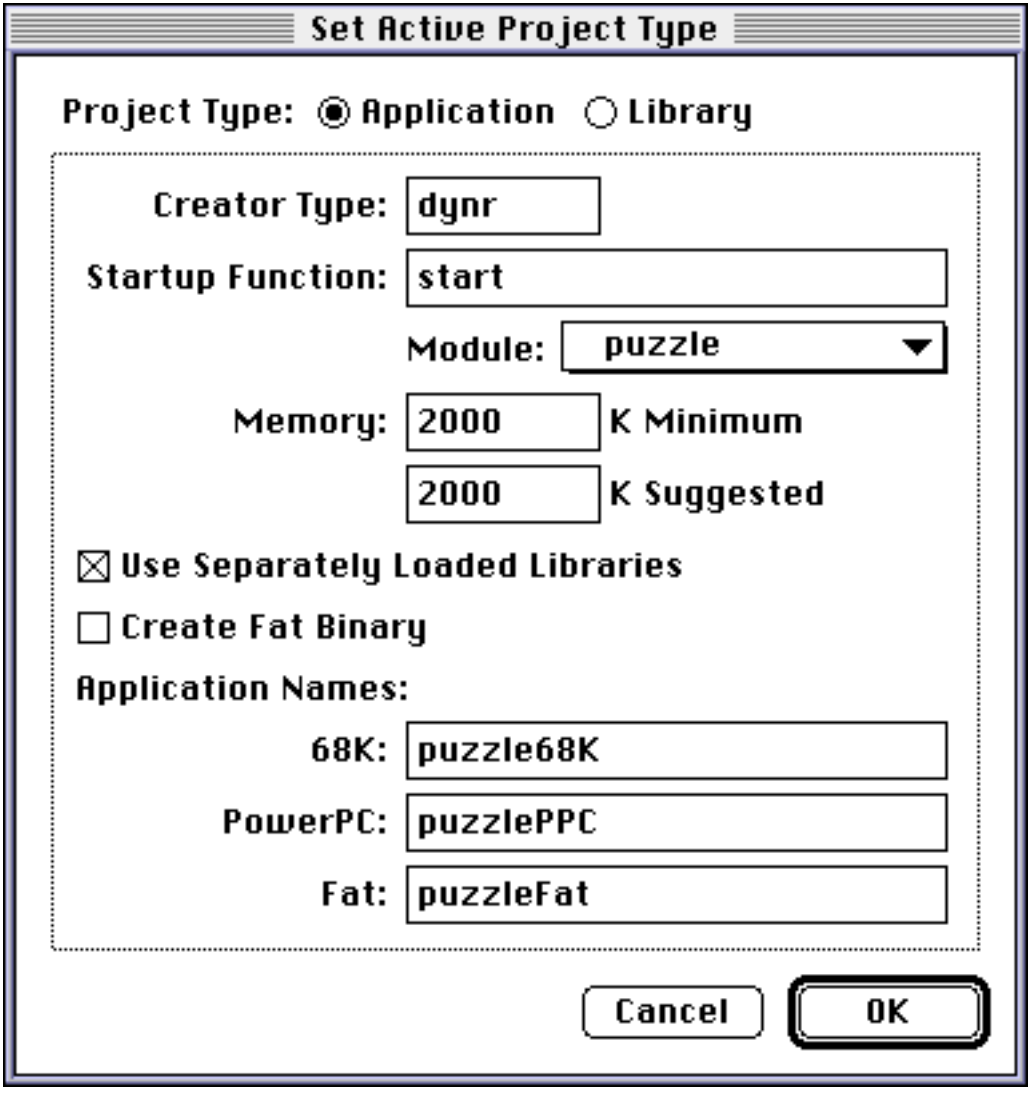
- **Note:** It sometimes happens that the application nub quits, and Apple Dylan doesn’t find out about it. In this case, there will still be a “Quit Application Nub” command on the “Project” menu, even though no application nub is running. To let Apple Dylan know it’s not connected, choose the “Quit application nub” menu-item, confirm that you want to disconnect, and then cancel the request to find the nub.
- **Note:** If you ever get into a state where the application nub is not responding and you cannot get it to quit, run the “Quit Application Nub” application in the “application nub” folder.

Creating a stand-alone application

There are two steps to creating a stand-alone application in Apple Dylan. You first specify some parameters of the application, such as its signature and startup function in the “Set Project Type…” dialog. You then use the “Create Application” command to actually build the stand alone application. What follows is a more detailed description of each of these two steps.

Specifying a project type

You specify project parameters via “Set Project Type…” on the “Project” menu. Choosing the command displays the following dialog:



All the information in the dialog is remembered as part of the project. It does not need to be set every time an application is created. You only need to bring up the dialog when you want to change or review its contents.

The “Set Project Type” dialog has the following fields:

- Project Type:
  A choice of “Application” or “Library.” Libraries are described in the following chapter. If you choose “Library” the remaining fields will be different, and are described in the next chapter.
- Creator Type:
  The creator ID to give the new application.
- Startup Function:
  The function to call to start up the application. Applications built using the framework, usually use the startup function `start`.
- Module:
  The module containing the startup function.
- Memory:
  Minimum and suggested memory configurations for the application.
- Use separately loaded libraries:
  Whether to bundle the libraries of any subprojects into the stand alone application, or leave them separate.
- Create Fat Binary:
  Whether to create a fat binary. If this option is not selected, “Create Application” will create an application suitable for the currently selected target architecture.
- Application Names:
  Names to use when creating 68K, PowerPC, and Fat applications. The applications will be saved in the project folder, using these names.

The project information is already properly filled out for the puzzle project, so you shouldn’t make any changes.

Create an application

After checking the dialog settings, you can choose “Create Application” from the project menu. A progress indicator will be shown as the application is being saved to the project’s folder.

- **Note:** Applications can only be built when disconnected from the nub. If you are connected, choosing “Create Application” will confirm with you that it is okay to disconnect from the nub before continuing.
- **Note:** If you save an application when the project’s folder is open in the Finder, the application may appear as a document rather than as an application. To convince the Finder it is an application, you must close and open the folder containing it, or moved it into a closed folder.

Once the application has been created, you can double click it to launch it.

Congratulations, you’ve created your first Apple Dylan application!

CHAPTER 3

Tutorial

CHAPTER 4

Libraries

Contents

This chapter explains the use of libraries and library files in Apple Dylan, both during development and during application delivery.

Overview

Libraries are a feature of the Dylan language. A library is a group of modules which are compiled and linked together. A library includes exported modules and unexported modules. Exported modules can be imported by other libraries, unexported modules can't be. The exports and imports of a library are specified by a `define library` definition.

Libraries can be used to divide applications into parts which are separately compiled and linked, and may be separately delivered to other developers and end users in the form of library files. In addition, the speed of many development operations can be greatly increased by pre-linking libraries. This process is described below.

Library files are a feature of Apple Dylan. A library file is the compilation product of a library, and is used to support separate delivery of the pieces of an application. The presence of library files in the right locations can also make some development activities faster.

For additional information on libraries as they relate to the Dylan language, see the Dylan Reference Manual. The remainder of this section describes libraries and library files as they relate to Apple Dylan projects.

Libraries, Projects, & Subprojects

Every Dylan project is associated with a library. The library consists of the modules in the project. You can supply the library definition (with `define library`), but if you don't, a default library definition will be calculated automatically. This default library will use the library of each subproject, and will not export any modules.

When a project has subprojects, each subproject is associated with its own library.

- **Note:** Library definitions should be compiled immediately after they are modified. Do this by typing enter or command-E while the blinking insertion bar is in the text of the library definition or the icon of the library definition is selected.

The `define library` definition is part of the source code of a project, and is usually placed in the Dylan-User module. A Dylan project cannot contain more than one `define library` definition.

When a library uses another library, the using library can import modules from the library being used. When project A has project B as a subproject, the library definition of project A will usually use the library of project B and import modules which are exported by the library definition of project B. This is declared by the `define library` statement.

- **Note:** A could use B without importing any modules from B if B adds methods to some generic functions and thereby extends the functionality of A. This can only happen if there is a common library which both A and B use.

Most programs written in Apple Dylan will contain at least three subprojects:

1. The Dylan subproject, which exports modules containing the standard Dylan language definitions and other modules containing Apple extensions to the Dylan language. Every project must contain this subproject.
2. The Framework subproject, which contains the Dylan Application Framework.
3. The Mac-Toolbox subproject, which imports some useful Mac Toolbox definitions using Creole and re-exports them. The Mac-Toolbox project is used by the Framework project, but you may want to use it in your projects as well.

- **Note:** You only need to use the Mac Toolbox library if you plan on accessing toolbox functions or types directly, rather than through the framework.

Library Files and Applications

When you deliver your application to an end user, you must ensure that the end user has access to all of the libraries used by the application. There are two ways you can do this. You can include the library in the application file itself,

or you can use separately loaded library files. Separately loaded library files appear as individual files on disk.

When you create an application using separately loaded library files, the project's library is included within the application but all subprojects' libraries are kept as separate individual files.

The choice between separately loaded library files and included libraries is an option of the "Project Type" dialog box which is accessed through the "Set Project Type…" command on the "Project" menu.

Including the libraries in the application file has the advantage of simplifying delivery and configuration for your end user. However, it has the disadvantage of make applications larger and using additional disk space for each different application which uses a library.

Using separately loaded library files reduces the disk footprint of applications by allowing a single library file to be shared by multiple applications.

- **Note:** A single Apple Dylan library file may be used by multiple applications, but each application will have its own copy of the library in RAM. At this time, Apple Dylan library files will not decrease the RAM footprint used by applications.

Library files are normally saved with the suffix ".dl", and have this Finder icon:

Library file search path

When you launch an Apple Dylan application, it looks for its libraries in the following locations, in the following order:

- First:
  in the application itself.
- Next:
  in the folder containing the application.
- Last:
  in the Extensions folder of the System folder of the startup volume.

Aliases are resolved as necessary in the process of searching for library files.

Library Names & Versions

When you save a library file, neither the name of the file nor its creator are significant. Applications which use the library locate it by its file type, and by the name of the library it contains as specified by the `define library` statement.

A library has a version and a minimum compatible version. These versions are set in the "Project Type" dialog box which is accessed through the "Set Project Type…" command on the "Project" menu.

A program built to work with version N of a library will accept any library file which has a version greater than or equal to N and a minimum compatible version less than or equal to N.

- **Note:** It's up to the programmer to know when to bump version numbers. Some guidelines are given in "Using the Apple Dylan Development Environment."

Pre-linking Libraries to Speed Development

You can pre-link libraries to greatly speed up the time it takes to launch an application or library under development.

When you launch the application nub and update, any code which is not in a pre-linked library will be linked and downloaded to the application nub. This can take a while, especially for large projects or projects with large subprojects (such as the Dylan Framework).

However, if there are pre-linked libraries for the project or any of its subprojects, these will be loaded into the application nub more quickly. Once the libraries are loaded, Apple Dylan checks to see if any code in the project or subprojects has changed since the libraries were linked. The next time you update, any code which has changed will be separately linked and downloaded on top of the pre-linked libraries. This process is much faster than not using pre-linked libraries at all.

When you've made a number of changes to a project, you can re-link the libraries, thereby saving the time to download the changed code.

- **Note:** Changed code will not be loaded until after all the libraries are loaded. This means that if you change code which causes a side-effect at start-up time, it will be executed twice: once when the library is loaded and again when the new version of the changed code is downloaded. When you change code that has such side-effects, you should re-link the library.

When a project's library is linked, the results are stored in a library model in the project folder. There will be a separate library model for each target architecture.

Using library files to speed development

Library files (".dl" files) hold pre-linked libraries in the form in which they can be delivered to end users. Library files can also be used to further increase the speed of installing libraries in the application nub. Loading a library from a library file is slightly faster than loading it from a library model.

The search path for library files during development is similar to the search path when applications are delivered: First the folder containing the application nub is checked, then the Extensions folder. If a library file is not found, the library will be loaded from the library model.

- **Note:** You can take advantage of this feature by placing the library files or aliases to the library files in your application nub folder.

Creating Pre-linked Libraries

The same mechanisms are used to create library models and library files. There are two such mechanisms:

1. When you create an application, the libraries of the project and its subprojects are brought up to date. That is, they are linked if they have been modified since the last time they were linked. The results are stored in a library model for each project and in a library file for each project, in each project's folder.

2. You can link the library for a project and its subprojects by making the project the active project and creating the library explicitly. To do this select "Set Project Type…" from the "Project" menu and choose the project type "library." Then choose "Create Library" from the "Project" menu. If any subprojects need to be brought up to date, you will be prompted for confirmation.

- **Note:** You cannot create a library or a stand-alone application while you are connected to the application nub. If you are connected to the application nub, selecting the "Create Application" or "Create Library" command from the "Project" menu will disconnect from the application nub before creating the application or library.

To make library files accessible during development, you must place them or aliases to them in the search path of the application nub. We recommend leaving each library file in its project's folder, and placing an alias to the library file in the application nub folder. This way new library files will replace old library files, and the aliases will refer to the new library files.

- **Note:** When you distribute an application that uses separately loaded libraries, make certain to distribute the required library files too. Instruct users that these library files can be kept either in the same folder as the application or in the Extensions folder in their System folder. Unlike Other system extensions, library files will not automatically be placed in the Extensions folder when they are dropped on the System folder. They must be explicitly placed in the Extensions folder.

Note: If you create a stand-alone application for your own use, you also have to move library files or aliases to library files from the application nub folder to the stand-alone application's folder or the Extensions folder.

Creating Your Own Projects

This section walks you through the process of creating a simple project which defines a library, and then creating a new project which uses the first project.

Create a new project

Begin by launching Apple Dylan. If Apple Dylan is already running, close any projects which you have open.

Create a new project by choosing “Project” from the “New” command on the “File” menu.

Rename the source folder

Click on the title of the source folder called “Untitled” in the lower left pane, and when it becomes editable, rename it to “Library and Modules”. (You can actually choose any name you want, but it’s conventional to use this name for the source folder holding your library and module definitions.)

Enter the library definition

The “Library and Modules” source folder contains one source record, which is displayed in the right hand pane. Click to the right of its icon to get a blinking insertion bar then enter the following source code:

```
define library arithmetic
  use Dylan;
  export simple-arithmetic;  // export the module
end library arithmetic;
```

- **Note:** Library definitions should be compiled immediately after modifying them. Do this by typing enter or command-E while the blinking insertion bar is in the text of the library definition or the icon of the library definition is selected.

Note that the unsaved status indicator appears next to the source record, and also next to the source folder and module which contain it.

Save the project

Before proceeding further, save your new project by selecting the “Save” Command from the “File” menu. You will be prompted for a name and location for the project. Save the project as “Integer Math”.

Enter the module definition

Create a new source record by choosing the “New Source Record” command from the “File” menu or by typing Command-N. Enter the following source code in the new source record:

```
 define module simple-arithmetic
  use Dylan;
  export plus, minus;  // export the names
end module simple-arithmetic;
```

Save the project again.

Create the remainder of the library

Compile the module definition. Note that a new module object appears in the upper left hand pane of the project browser.

The new module will be created with one source folder already in it. Rename this source folder “Plus and Minus.”

In the right hand pane, open the source record in the “Plus and Minus” source folder, and enter the following source code:

```
define method plus (number1 :: <integer>, number2 :: <integer>)
  => result :: <integer>;
  number1 + number2
end method plus;
```

Create a new source record in the “Plus and Minus” source folder and enter the following source code:

```
define method minus (number1 :: <integer>,
                     number2 :: <integer>)
  => result :: <integer>;
  number1 - number2
end method minus;
```

Save the project.

Finish creating the library

Select the "Set Project Type…" command from the "Project" menu and set the project's type to "Library". This dialog also lets you set version numbers for the library, names to use when saving the library, and a creator for the library file.

Close the dialog and choose the "Create Library" command from the "Project" menu to link the library and create a library file.

Close the "Integer Math" project.

Create another project

Create a new project and give it a "Library and Modules" source folder using the steps outlined above.

Click in the source record in the right hand pane and enter the source code:

```
define library integer-math-user
  use Dylan;
  use arithmetic;  // use the library
  export ones;     // export the module
end library integer-math-user;
```

Create a new source record, and enter the source code:

```
define module ones
  use Dylan;
  use simple-arithmetic;  // use the module
  export plus1, minus1;   // export the names
end module ones;
```

Save the project as “Integer Math User”.

Add a Subproject

Choose the “Add to Project…” command from the file menu. Add the “Integer Math” project. To do this, select the file “Integer Math.π” from the “Integer Math” folder.

Choose the “Update” command from the “Project” menu.

You’ve now created a project which uses a subproject and imports definitions from that subproject.

Define Some Functions

Select the newly created module “ones” in the upper left hand pane. Rename the untitled source folder in `ones` to “Addition”.

Add the following definitions to the “Addition” source folder:

```
define method plus1 (number :: <integer>)
  => result :: <integer>;
  plus(number, 1)
end method plus1;

define method minus1 (number :: <integer>)
  => result :: <integer>;
  minus(number, 1)
end method minus1;
```

Save the project.

Test the project

To test the project:

Launch the application nub (command-K).

Update the project (command-U). The "update on launch" preference lets you skip this step.

Set the listener's module to "ones."

Type some expressions into the listener. The ones module has access to the code it defines, as well as code imported from the simple-arithmetic and Dylan modules.

```
> plus1(100)
101
> minus1(77)
76
> plus(50, 50)
100
> 77 - 55
22
>
```

CHAPTER 4

Libraries

CHAPTER 5

The Application Nub

Contents

This chapter explains how to use different configurations of the development environment and application nub, including two-machine development.

Running a Program

To execute the code in a project, you must run an application nub and load the code into the nub.

Apple Dylan supports both one-machine and two-machine development configurations. In the one-machine configuration, both Apple Dylan and the application nub reside on the same machine. In the two machine configuration they reside on different machines connected by AppleTalk.

The steps for using the application nub are different for the one-machine and two-machine cases. In the one-machine case, the nub is launched and connected to the runtime with one command from Apple Dylan. In the two machine case, you launch the nub manually and than connect to it.

- **Note:** Regardless of how it is started up, the application nub will be launched with the amount of memory specified for it in the "Get Info" box of the Finder. When you are working on large projects, or projects which include the Dylan interface builder, you should make sure to increase the preferred memory size of the application nub.

Selecting Launch Preferences

The "Preferences" command on the "Edit" menu includes a panel for application nub preferences.

- Launch Application Nub on Project Activation
  If this preference is selected, the application nub will be launched when a project is opened, or when a new project is made active. (Active and inactive projects are described in "Using the Apple Dylan Development Environment.")
- Update when Application Nub is Launched
  If this preference is true, then an "Update" command is issued whenever the development environment connects to the application nub.

- Development Mode
  This preference let's you choose between one-machine and two-machine development. If two-machine development is chosen, you can also fill in the name of the runtime machine (the machine on which the application nub will be run). This is the name of the Macintosh as given in the "Sharing Setup" control panel.

One Machine Development

To start up and connect to the application nub in the one machine configuration, you choose "Launch application nub" from the "Project" menu after opening your project.

- **Note:** You can also use the "Run" command on the "Project" menu, which will launch the nub if it's not already running, update, and call the project's startup function.

The search path for the application nub begins with the folder of the active project, and then proceeds to the folder "Apple Dylan Files:application nub:". The application nub is identified by name.

If the application nub can't be found, either because it is missing or an alias cannot be resolved, an alert will be presented asking you to locate an application nub. You are presented with three choices:

- Launch
  Lets you locate an application nub on disk using the standard file dialog. The nub will be launched and connected.
- Choose
  Lets you locate a running application nub using the PPC browser. This nub can be on the same machine or on a different machine. It will be identified by its application type, not name.
- Cancel
  Lets you cancel the operation.

Two Machine Development

Setting up for two-machine development involves the following steps:

- Select two-machine development and enter the runtime machine name in the preferences panel.
- Turn on "program linking" in the "Sharing Setup" control panel of the runtime machine. Also on the runtime machine, use the "Users & Groups" control panel to set up a user which allows program linking.
- Copy the application nub and any library files you will be using to the runtime machine. All of these should be in the same folder, or the library files can be in the Extensions folder of the runtime machine. The library files to copy include the Dylan library file, the Framework library file, and the Mac Toolbox library file.
- Using personal Appleshare, mount the volume containing your project on the runtime machine. The application nub needs this volume mounted so it can access any resources in the project folder.

Macintosh program linking does not allow one Macintosh to launch an application on another Macintosh. You must launch the application nub by hand, by double-clicking it in the Finder.

Once you're set up for two machine development, you connect by performing the following steps:

- Launch the application nub on the remote machine.
- Choose the "Launch Application Nub" from the "Project" menu of the development environment.
- Respond to the ensuing dialog by entering the appropriate user name and password.

Once you're connected, development operations can be performed just as in one-machine development.

Disconnecting from the Application Nub

When you close a project, the application nub will automatically be quit.

If you want to disconnect from the application nub without closing the project, you can do this manually by choosing “Quit application nub” from the “Project” menu. This command will only work if the application nub is in a state where it can receive commands from Apple Dylan (that is, if it is paused).

If the application nub gets stuck open, you can force it to quit by running the “Quit application nub” program provided with Apple Dylan.

Note: If you quit the application nub this way (or if you simply quit the application nub by using the “Quit” menu-item in the application you are developing), Apple Dylan will not know that the application nub has been quit. The “Quit application nub” command will still appear on the “Project” menu. Choosing this command will spuriously ask you to locate the running application nub. You can simply cancel the dialog containing that request.

Connecting to Running Applications

If you create a stand-alone application with Apple Dylan and the application encounters an error while it is running, you can connect the application to the development environment for debugging.

When the application encounters an error that isn’t handled, it will present a dialog box offering to quit to the Finder. To debug the application, hold down the option key and click on the “Quit” button of the dialog box. The dialog will go away but the application won’t quit. Then return to the Finder, launch Apple Dylan, open the project which was used to create the application, and choose “Tether to Application” from the “Project” menu.

Once you have tethered, you can debug just as if the application had been launched from inside Apple Dylan. If you repair the error situation and continue from an appropriate restart, you can untether by selecting the “Untether from Application” command on the “Project” menu.

CHAPTER 5

The Application Nub

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Line art was created using Adobe Illustrator™ and Adobe Photoshop™.

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.

WRITER
Andrew Shalit

ILLUSTRATOR
Steve Strassmann

PRODUCTION EDITOR
Lorraine Findlay, JoAnne Smith

Special thanks to Ricardo Gonzalez and Russ Daniels for their ongoing support.

Apple Dylan has been the work of many hands over many years. Through its evolution as a language design, research project, and now product effort, it has benefited from the contributions, support, and helpful comments of too many people to list in this small space. You know who you are! Thank you for your help in making the world a better place for programmers. Carry the flame forward!