Harlequin Dylan

Release and Installation Notes

Version 2.0 Beta

harlequin™

http://www.harlequin.com/

Europe:

Harlequin Limited
Barrington Hall
Barrington
Cambridge CB2 5RG
UK

telephone +44 1223 873 800
fax +44 1223 873 873

North America:

Harlequin Incorporated
One Cambridge Center
Cambridge, MA 02142
USA

telephone +1 617 374 2400
fax +1 617 252 6505

Asia Pacific:

Harlequin Australia Pty. Ltd.
Level 12
12 Moore Street
Canberra, ACT 2601
Australia

telephone +61 2 6206 5522
fax +61 2 6206 5525

Contents

1 Editions of Harlequin Dylan 5

Harlequin Dylan Personal Edition 6
Harlequin Dylan Professional Edition 7
Harlequin Dylan Enterprise Edition 7

2 Documentation 8

3 Installing Dylan 9

System requirements 9
Installing Microsoft Internet Explorer 11
Upgrading to Harlequin Dylan version 2.0 from an earlier version 12
Download installation procedures 12
CD-ROM installation procedure 13
License information 14
Starting Harlequin Dylan 14

4 Reporting Bugs and Troubleshooting 14

5 What's New in This Release? 15

Profiling 15
The Dylan Object Oriented Database (DOOD) 15
Enhanced foreign keyboard support 16
Source code control 16
Improved Dylan libraries 16

Compiler improvements 16
Environment enhancements 17
Debugging enhancements 17
New examples 18

6 Libraries 18

Harlequin Dylan library structure changes 18
New UDP Sockets library 19
New Win32-MULTIMEDIA interface library 20
New OpenGL interface libraries 20

7 Software Issues 20

Other development environment changes 20
Limitations to new source control interface 21
The linker 21
Using resources in a project 22
Compiler notes 22
Development environment problems 26
C-FFI library 27
Non-local exits 27
OLE library macros 27
SQL-ODBC library 28
Harlequin Dylan’s CORBA® features 30
ORB™ runtime 31

8 Supplemental Documentation 35

Profiling 35
The Dylan Object Oriented Database 40

Harlequin Dylan Release and Installation Notes

These notes provide important introductory information as well as corrections and supplements to the main Harlequin Dylan documentation set.

The release notes consist of:

- Installation instructions.
- Information on reporting bugs.
- Details of problems in and late changes to the product software.

1 Editions of Harlequin Dylan

Harlequin Dylan comes in three editions: the Personal Edition, the Professional Edition, and the Enterprise Edition. In general these notes apply to all editions. However, items that apply only to one are flagged as **[Professional edition]**, **[Enterprise edition]** or **[Personal edition]**. Where there might be ambiguity after two such paragraphs, a return to general material is signaled by **[All editions]**.

If you enjoy using the Harlequin Dylan 2.0 Beta, you may be interested in the 1.2 Professional and Enterprise Editions, including support for CORBA, COM and networking code. See the following sections for information on these features and how to contact Harlequin.

1.1 Harlequin Dylan Personal Edition

The Harlequin Dylan Personal Edition includes everything needed to produce standalone applications and DLLs for the Windows 95, Windows 98, and Windows NT operating systems.

1.1.1 Personal Edition Library pack

We have collected some specialized libraries (providing extended features) into a separate installer, Library Pack 1. To use these libraries you must also install the Personal Edition.

Library Pack 1 contains seven C-FFI libraries providing interfaces to Win32 API features: Win32-GL (OpenGL), Win32-GLU (OpenGL), Win32-Multimedia, Win32-DDE, Win32-Rich-Edit, Win32-Shell, and Win32-Version. (The latter four of these libraries were in the standard distribution of the 1.0 Personal Edition.) To run the OpenGL demos, the library pack must be installed.

Like the documentation installer and the Personal Edition itself, Library Pack 1 is available from the following address.

`<URL:http://www.harlequin.com/devtools/dylan/>`

See Section 6, “Libraries” for more library information.

1.1.2 Personal Edition C interoperability

The Personal Edition of Harlequin Dylan supports only the GNU linker. The Microsoft linker cannot be used with the Personal Edition.

In addition, the Personal Edition has limited support for linking to C libraries. Only the standard Windows libraries as supplied with the Personal Edition are supported. (See the **`Tools\i386-cygwin32\lib`** folder in the top-level Harlequin Dylan installation folder.) These libraries are supplied in GNU archive format. The Personal Edition does not support linking to other libraries written in C, including C libraries you may have written yourself, with any C compiler.

In order to use the Microsoft linker or link to arbitrary C libraries, or to include C or C++ source files in your project, you must use the Professional or Enterprise Editions of Harlequin Dylan.

1.2 Harlequin Dylan Professional Edition

The Harlequin Dylan Professional Edition is built on the same compiler and integrated development environment as the Personal Edition. It adds a number of features to support those who wish to use Dylan for more serious Windows development:

- Support for COM, OLE, and ActiveX.
- Full support for linking to libraries written in C.
- Support for network programming through sockets.
- ODBC connectivity.
- Just-in-time debugging.
- Over 300 pages of printed documentation.
- 60-day free “getting started” support.

Harlequin Dylan Professional Edition is available for sale on the Harlequin Web site at the following address:

`<URL:http://www.harlequin.com/devtools/dylan/>`

You can also buy it by telephone by calling one of the following numbers:

+1-617-374-2521 (US)

+1-888-884-8871 (US Toll-Free)

+44-1223-873883 (UK)

1.3 Harlequin Dylan Enterprise Edition

The Harlequin Dylan Enterprise Edition includes all the features of the Professional Edition, and:

- Support for the CORBA distributed object standard.

- Remote development technology, allowing you to debug your applications across a network.

Further features are in development.

2 Documentation

The 2.0 Beta release includes:

- A complete 2.0 Beta documentation set in PDF, HTML, and HTML Help formats
- The book *Dylan Programming* in HTML and HTML Help formats (made available on-line for the first time)

You can browse any of the HTML documentation on-line (from the CD-ROM) or install it onto your machine. To install all of the documentation from the CD-ROM, see Section 3.5 on page 13.

The Harlequin Dylan 2.0 Beta documentation set is provided in three electronic forms:

- An HTML-based version of the documentation that uses Microsoft's HTML Help technology. HTML Help provides a global text search facility as well as a master table of contents and index for the entire documentation set. To use the HTML Help version of the documentation, you must already have Microsoft Internet Explorer 3.02 (or a later version) installed on your computer.
- A standard HTML 3.2 web of HTML files (one web per manual). You can view this documentation in any HTML 3.2-compliant browser, such as Netscape Navigator 4. It is exactly the same as the HTML Help documentation except that it does not support global text searching or provide a master table of contents and index.

- A set of PDF files (an Adobe Acrobat format). This format is suitable for printing. Each manual in the documentation set is presented in a separate PDF file, available from the Start menu under **Start > Programs > Harlequin Dylan > Printable Documentation**.

  To view and print these files you will need a copy of Adobe's Acrobat Reader 3.0 application. You can download a free copy from Adobe's web site.

[Professional and Enterprise Editions] The Professional Edition installer includes all three forms of the documentation. You can choose which ones you want during installation.

[Personal Edition] The Personal Edition installer does not include any documentation. To install documentation you must download the separate documentation installer from this address:

```
<URL:http://www.harlequin.com/devtools/dylan/>
```

3 Installing Dylan

This section outlines the and system requirements and other software needed to install Harlequin Dylan and then the actual installation procedures. Harlequin Dylan is available both on CD-ROM and as a download from the Harlequin Web site.

3.1 System requirements

Harlequin's implementation of Dylan is targeted at the 32-bit Windows operating systems (Windows 95, 98, and Windows NT) running on a 100MHz Intel Pentium (or equivalent) processor.

We recommend that you have:

- A minimum of 32 MB of RAM. 64 MB or more is preferred.
- A minimum of 128 MB of virtual memory. 160 MB or more is preferred.

	Personal	Professional	Enterprise	Documentation
Target folder (permanent space)	95 MB	150 MB	160 MB	7.5 - 30 MB
Temporary space for Web downloads	32.5 MB	45 MB	47.5 MB	17.5 MB

Table 1

Note: If the **`TMP`** environment variable is set, it will be used as the temporary folder. If it is not set, then the location for temporary files depends on which operating system you have. On Windows NT, it will be the *Windows folder*, which is the value of the **`windir`** environment variable. On Windows 95 and 98 machines, the current directory is used.

To avoid problems during Web download and installation, make sure that the temporary directory your system is going to use has sufficient free disk space. If necessary, set the **`TMP`** environment variable to a directory with adequate space.

1. Open a command prompt (**Start > Programs > Command Prompt** on NT or **Start > Programs > MS-DOS Prompt** on 95/98).
2. Type **`set TMP=`*****path*** where *path* is the path to the temporary directory you want to use (for example, **`d:\tmp`**).
3. **[Personal Edition]** Type **`cd`** *directory* where *directory* is the directory into which you downloaded the installer for the Harlequin Dylan Personal Edition. Then type **`start personal`**.

   [Professional and Enterprise Editions] Type **`start`** ***drive*****`:\browser`** where *drive* is the drive letter of your CD-ROM drive.
4. Follow the installation procedure described in Section 3.5.

3.1.1 Updating Windows 95 socket 2

Windows 95 users may need to update their winsock2 DLL files in order to run Harlequin Dylan. Harlequin Dylan 2.0 uses newer versions of the DLLs than Harlequin Dylan 1.2. If the winsock2 DLLs are out of date, users will see a dialog about missing DLLs ws2_32.dll and ws2help.dll. Or users may see a “socket error” message from Harlequin Dylan.

The latest winsock2 can be downloaded from:

```
www.microsoft.com/windows95/downloads/default.asp
```

Click on the “Windows Socket 2 Update” link near the bottom of that page.

3.2 Installing Microsoft Internet Explorer

To use Harlequin Dylan’s HTML-Help–based on-line documentation, with its master table of contents, master index, and full text search feature, you need to have installed Microsoft Internet Explorer 3.02 or later. You can download Internet Explorer from:

```
<URL:http://www.microsoft.com/windows/ie/download/default.asp>
```

If you choose the HTML Help documentation during installation, the installer checks to be sure you have the correct version of Internet Explorer. If it thinks you do not, a dialog appears with instructions.

On Windows NT, you may need to install Internet Explorer with administrator privileges.

If you experience problems with installing Internet Explorer, please make sure that the following registry entry exists.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Internet Explorer
MkEnabled = "Yes"
```

You can use the registry editor to check this. If the registry entry does *not* exist it is an indication that an Internet Explorer subsystem known as the InfoTech protocol has not installed properly. In this case, if you are using Windows NT, try reinstalling Internet Explorer 3.02 with administrator privileges.

3.3 Upgrading to Harlequin Dylan version 2.0 from an earlier version

If you are upgrading your copy of Harlequin Dylan from version 1.1 or 1.2 to version 2.0, you need to uninstall the older version first. Despite doing this your compiler and environment settings—such as your choice of linker and editor options—will be carried forward to version 2.0.

[Personal and Professional Editions]

To uninstall the older version:

1. From the Windows **Start** menu, choose **Settings > Control Panel**.
2. Choose **Add/Remove Programs**.
3. Select **Harlequin Dylan** from the list of programs.
4. Click **Add/Remove…**.

The uninstaller takes over from this point. If you have modified or created any files under the top-level Harlequin Dylan installation folder, the uninstaller leaves the files alone and reports that some files were not removed. You must remove any such files by hand, or move them elsewhere if you prefer.

(Note that if you have built any of the example projects, the files and folders created during the builds will not be removed automatically.)

5. Remove any remaining files and folders from the top-level installation folder.
6. Install version 2.0 either from the CD-ROM **[Professional and Enterprise Editions]** or by double-clicking on the installer executable **[Personal Edition]**.

3.4 Download installation procedures

All three editions of Harlequin Dylan and the Harlequin Dylan documentation are available for downloading from the Harlequin Web site at the following address:

`<URL:http://www.harlequin.com/devtools/dylan/>`

To download and install Harlequin Dylan 2.0 Beta:

1. Download the appropriate installer from the Harlequin Web site.

   If you choose Save to Disk, the browser will let you choose where it copies the downloaded files. You then double-click on the installer executable to begin installation.

   If you choose to Run the software where it is, the browser copies the downloaded files to a temporary directory, validates the digital signature, and starts the installer automatically.

2. The installer starts up and leads you through a series of standard installation dialogs. The installer will give you the option of installing the documentation along with the product.

   If you decide to install the documentation later, follow the installation procedure again and at this step click the appropriate **Install Documentation** button. See Section 2, “Documentation” for details.

3.5 CD-ROM installation procedure

To install Harlequin Dylan version 2.0 Beta from a CD-ROM:

1. Mount the Harlequin Dylan 2.0 Beta CD-ROM on your drive.

   The Welcome to Harlequin Dylan dialog appears.

2. Click **Install**.
3. Click **Harlequin Dylan Version 2.0 Beta**.
4. Click the appropriate edition: Personal, Profession, or Enterprise.
5. The installer starts up and leads you through a series of standard installation dialogs. The installer will give you the option of installing the documentation for the Personal Edition along with the product.

   If you decide to install the documentation later, follow the installation procedure again and at this step click the appropriate **Install Documentation** button. See Section 2, “Documentation” for details.

3.6 License information

The installer will prompt you for a serial number and license key. This will be new numbers for Harlequin Dylan 2.0 Beta, not the numbers used for earlier versions.

Professional serial numbers start with **`HDPRO`**; Enterprise serial numbers start with **`HDENT`**.

The 2.0 Beta licenses expire. Runtimes expire November 15. Personal edition licenses expire September 1.

3.7 Starting Harlequin Dylan

When the product installation is complete, you can start Harlequin Dylan by choosing **Start > Programs > Harlequin Dylan > Dylan**.

4 Reporting Bugs and Troubleshooting

Beta customers can contact Harlequin Support and report bugs and problems through email to
`dylan-support@harlequin.com`.

New users may like to explore the Harlequin Dylan Knowledgebase, which provides introductory information, examples, and troubleshooting advice. You can access the Knowledgebase in the Support area of our Web site:

`<http://www.harlequin.com/devtools/dylan>`

You can subscribe to the **`info-dylan`** mailing list by sending mail to

`info-dylan-request@harlequin.com`

or, for the digest version,

`info-dylan-digest-request@harlequin.com`

Please contact Harlequin Support in the following circumstances:

- You need assistance with installing or running Harlequin Dylan.
- You encounter a bug in the Harlequin Dylan software or documentation and wish to submit a bug report.

- You have any comments or suggestions for improving the Harlequin Dylan software or documentation.

If you are opening a new support call, please provide the following information:

Serial number	Located on the back of the CD envelope.
Machine & OS	The computer and OS version.
Dylan Version	The version of Harlequin Dylan you are using.
Description	A full description of the bug or question.

If you can, please also provide:

Repeat by	If a bug, step-by-step action to reproduce it.
Test	A small test case helps us respond more quickly.

5 What's New in This Release?

The Harlequin Dylan 2.0 Beta contains the following improvements over the 1.2 release.

5.1 Profiling

As of the 2.0 Beta release, Harlequin Dylan supports profiling of *any* kind of application. Using the Dylan profiling tool you can significantly improve the performance of any application. You can opt to profile an application either at a fixed rate or according to memory allocation. See the documentation for this feature in Section 8.1 on page 35.

5.2 The Dylan Object Oriented Database (DOOD)

DOOD is a simple mechanism for storing arbitrary objects and lazily loading them. DOOD supports:

- pay as you go
- incremental loading

- flexible proxies
- non-corrupting commits

For documentation on this feature, see Section 8.2 on page 40.

5.3 Enhanced foreign keyboard support

Foreign keyboard support is enhanced in this release.

5.4 Source code control

The new source control interfaces with Microsoft's Visual SourceSafe.

5.5 Improved Dylan libraries

- Limited collections support

  Supports better error-checking and more efficient code
- Improved Windows look-and-feel for DUIM

  Improvements to focus management and keyboard navigation

5.6 Compiler improvements

- Faster generated code
- Smaller executables
- Improved interactive execution:
  - restarts for runtime errors
  - graceful resolution of undefined bindings

  This supports smoother interaction and more a dynamic programming style.
  - More, and improved, warnings

5.7 Environment enhancements

- Command line version for scripting, **File > Command Line** from the main window
- Console environment (`console-dylan.exe`) for batch compiling *and* complete interactive development
- Project window's Warnings page shows the range of problematic code lines, when applicable
- Editor enhancements:
  - more editor options: using Emacs-style incremental search as the default search (in Emacs mode only) and using the clipboard as well as the "kill ring" for Cut, Copy, and Paste
  - highlighted range of text: the range of code lines corresponding to a compiler warning is highlighted when viewed in the editor
  - name completion in Emacs mode: use `Meta+/` to complete a name and cycle through the matches
  - macro expansion: place your cursor in a macro and use **Project > Macroexpand Selection** to see what the macro will do, use **Edit > Undo** (or the toolbar / keyboard equivalents) to return to the original buffer

5.8 Debugging enhancements

- Redesigned debugger window:
  - one debugger window for debugging multiple threads
  - new menu items for selecting threads
  - drop-down list for easy selection of options to filter the frames displayed in the stack backtrace
  - messages now appear in the interaction pane
  - menu item for hiding the context pane
- New support for exporting bug reports and compiler warnings
- Stepping is much more reliable

- Function call tracing
- Raw memory and register browsing

5.9 New examples

- The ASP View example – an example program that demonstrates a WWW Server-side Automation component using the Microsoft ASP protocols. It also demonstrates the use of type libraries in a Dylan project.
- OLE Bank Client and Bank Server
- The Win32 Scribble example, which demonstrates how to print.
- The Airport example, which is documented in the book *Dylan Programming* which is supplied in the form of HTML Help. See Section 2 on page 8.
- Towers of Hanoi
- DUIM Editor
- CORBA Network Pente

6 Libraries

This section describes the new libraries in Harlequin Dylan since version 1.0.

6.1 Harlequin Dylan library structure changes

The following new libraries have been added to Harlequin Dylan. These libraries provide a new layout of the existing Harlequin Dylan libraries. They are:

Collections	Exports all modules that the Table-Extensions library exports. Complete list of Table-Extensions modules: Table-Extensions. (see next page)

IO	Exports the modules that the Streams, Print, Standard-IO, Format-Out, and Format libraries export. Complete list of Streams modules: Streams, Streams-Internals Complete list of Print modules: Print, PPrint, Print-Internals Complete list of Standard-IO modules: Standard-IO Complete list of Format-Out modules: Format-Out Complete list of Format modules: Format, Format-Internals
System	Exports the modules that the Date, File-System, and Operating-System libraries export. Complete list of Date modules: Date Complete list of File-System modules: File-System Complete list of Operating-System modules: Operating-System

The component libraries of these new libraries are still available directly but are now deprecated, and will be removed in a later release. You should update your libraries' **`define library`** (but not **`define module`**) declarations to use these new libraries at the first opportunity.

6.2 New UDP Sockets library

[Professional and Enterprise Editions]

UDP socket support has been added to the Network library. This library provides TCP server and client sockets and an Internet addressing protocol.

This library is only available in the Professional and Enterprise Editions of Harlequin Dylan.

6.3 New Win32-MULTIMEDIA interface library

Since version 1.1 there has been a new library called Win32-Multimedia. It consists of C-FFI declarations corresponding to the Windows header file **`mmsystem.h`** and the library **`winmm.lib`**.

The library includes support for wave form sound, MIDI, auxiliary audio, mixer, timer, joystick, MCI, and multimedia files.

Win32-Multimedia is available pre-installed in the Professional Edition. Personal Edition users can get it from Library Pack 1.

6.4 New OpenGL interface libraries

Two new libraries providing OpenGL support were introduced in version 1.1. They are Win32-GL and Win32-GLU.

Win32-GL consists of C-FFI declarations corresponding to the Windows header file **`gl.h`** and the library **`opengl.lib`**. Win32-GLU also consists of C-FFI declarations, corresponding to the Windows header file **`glu.h`** and the library **`glu32.lib`**.

The Harlequin Dylan Examples dialog (**Help > Open Example Project…**) contains a new OpenGL category, containing example programs created using these OpenGL libraries.

Win32-GL and Win32-GLU are available pre-installed in the Professional Edition. Personal Edition users can get them from Library Pack 1.

7 Software Issues

This section describes exceptions from documented behavior that exist in this version of the Harlequin Dylan software. In some cases, the descriptions are of bugs that will be fixed in the future. In other cases the descriptions are of last-minute changes that did not make it into the main documentation.

7.1 Other development environment changes

Since version 1.1, commands from the **Application > Breakpoints** menu of the project window, editor, and debugger have been moved to the **Application** and **Go** menus.

The **Project > Settings...** dialog's Link page now allows you to set a base address value in hexadecimal. Hex values must use Dylan's **#x** prefix.

The **Go** menu is now available from the project window, browser, editor, and debugger. The old **Application > Breakpoints > Browse All Breakpoints** command is now available here as **Go > Breakpoints**. The menu also contains two new items, **Go > Module** and **Go > Library. Go > Module** offers a submenu of modules defined in the project. **Go > Library** offers a submenu of libraries defined in the project. Selecting a module or library opens the browser on its definition.

7.2 Limitations to new source control interface

The following limitations apply to the new source control interface with Microsoft's Visual SourceSafe:

- The SourceSafe login dialog displayed by the Harlequin Dylan environment does not include a list of the known Visual SourceSafe databases; the user must enter the correct database name from memory. Additionally, there is no mechanism to select a database that is not yet known to Visual SourceSafe.
- Only one SourceSafe database can be used in a single invocation of the Harlequin Dylan environment.
- The Harlequin Dylan environment's initial guess of the SourceSafe project for a file is almost certainly incorrect.
- The Show History command is not implemented in Beta 2.
- The Show Differences command is not implemented in Beta 2.

7.3 The linker

The GNU linker is used by default in all editions of Harlequin Dylan.

[Professional and Enterprise Editions] If you have the Professional or Enterprise Edition, you can also choose to link Dylan applications with the Microsoft Visual C++ linker (version 4.2 or 5.0; preliminary tests indicate that the 6.0 linker will also work). Note also that if your project includes C or C++ sources, you *must* use the Microsoft Visual C++ linker.

To use the Microsoft linker, you must own a copy of Visual C++, because the linker is not part of Harlequin Dylan. The Microsoft Platform SDK also includes the linker executable, but does not include all the DLLs needed to run the linker. Attempting to choose the Microsoft linker without all the DLLs results in an error message such as:

```
The dynamic link library MSVCP50.DLL could not be found in the
specified path path
```

As well as installing the linker, you need to set some environment variables to be able to use it. Visual C++ includes the script `VCVARS32.BAT` which sets the variables to the required values. On Windows 95 and Windows 98, edit `AUTOEXEC.BAT` to make it run `VCVARS32.BAT` on startup. On Windows NT, you can copy the settings from `VCVARS32.BAT` by hand into the Environment page of the Control Panel's System category.

Once you have installed the Microsoft linker, you can use it either by explicitly adding `/linker microsoft` to your `console-dylan` command line, or by making it the default linker for Harlequin Dylan. To make the Microsoft linker the default linker, choose **Options > Environment Options...** in the Harlequin Dylan environment's main window. In the dialog that appears, select the Build page and then choose "Microsoft linker" in the "Link object files using" group.

7.4 Using resources in a project

In order to compile resource files, Harlequin Dylan depends on the Microsoft Platform SDK build environment being installed. If you do not have the Platform SDK, go to

```
http://msdn.microsoft.com/developer/sdk/sdktools.asp
```

and click "Download the Microsoft Platform SDK build environment (10.0 MB)".

7.5 Compiler notes

This section describes known problems and limitations of the Harlequin Dylan compiler.

7.5.1 Forward references

There are no checks on unbound variables or constants during execution of top level initialization code, including class definition, when an application or DLL is started. Forward references to unbound variables or constants in interactive development mode compilation can cause runtime crashes.

A macro's definition must be processed before a call to the macro can be parsed. Forward references to macros are not permitted or supported in any compilation mode.

7.5.2 Potential interactive redefinition crashes

Interactive redefinition is only possible for libraries compiled in interactive development mode. What you can do without risk:

- Redefine methods.
- Redefine a constant or variable to have a new value.

Unexpected or unpredictable results can occur if any of the following redefinitions happen:

- Redefinitions that change the type of a binding, for example changing a class to a function or a constant to a variable.
- Redefinitions from variable to thread variable.
- Redefinitions of modules or libraries.

7.5.3 Incremental redefinition of modules and libraries

Redefinition of modules or libraries does not properly force all necessary recompilation. If you are having trouble with compilation in the environment, you may want to try the batch compiler or **Project > Clean Build**.

7.5.4 Integer implementations

Instances of the class **`<integer>`** are represented using 30 bits. They can have values bounded in the following range:

```
$minimum-integer <= x <= $maximum-integer
```

The value of **`$minimum-integer`** is -2^29 and the value of **`$maximum-integer`** is 2^29 - 1.

These constants are exported from the Dylan module of the Dylan library. Though not listed in the current edition of the Dylan Reference Manual, they are now official extensions to the language.

If you want to represent integer values outside this range, you should use the Big-Integers module of the Big-Integers library. Some notes about Big-Integers follow.

- Big-Integers does not provide arbitrary precision integers. Only 64-bit integers are supported. They are represented by the class **`<double-integer>`**, a subclass of **`<abstract-integer>`**.
- Integer literals larger than **`$maximum-integer`** are represented using **`<double-integer>`**.
- Multiplication does not always signal overflow correctly when using the Big-Integers library.
- All forms of division (**`floor/`**, **`truncate/`**, **`round/`**, **`ceiling/`**) using Big-Integers will signal an error if both values are outside the range:

  `$minimum-integer <=` *x* **`<= $maximum-integer`**

  So, the following expression works:

  ```
  floor/(2 ^ 32, 10)
  ```

  while this one fails:

  ```
  floor/(2 ^ 35 + 5, 2 ^ 32 - 1)
  ```

7.5.5 Limited type support

Limited collection support has been expanded in this release to support instantiable limited collection types of certain collections (vectors, arrays, stretchy-vectors, and tables) on any element-type.

Certain limited collections over a few element-types are treated specially and return more efficient representations. Browsing currently does not show the elements of these built-in limited collections. The size and dimensions key-words of limited are unsupported. Uninstantiable limited collection types are unsupported. (See the list of uninstantiable limited collection types in the *The Dylan Reference Manual,* page 129.) The following is a list of the limited types supported in 2.0 Beta:

```
define method limited
    (class :: subclass(<collection>), #key of, size, dimensions)
  => class;

define method limited
    (class == <string>, #key of, size)
  => class
define method limited
    (class == <range>, #key of, size)
  => class
define method limited
    (class == <deque>, #key of, size)
  => class
define method limited
    (class == <stretchy-vector>, #key of)
  => limited-stretchy-vector(of)
define method limited
    (class == <vector>, #key of, size)
  => limited-vector(of)
define method limited
    (class == <array>, #key of, size, dimensions)
  => limited-array(of)
define method limited
    (class == <table>, #key of, size)
  => limited-table(of)

where limited-vector returns builtin class representations for
vectors of <byte>, <double-byte>, <machine-word>, <integer>,
<single-float>, and <double-float> and otherwise returns an
instantiable limited collection type.

where limited-array returns builtin class representations for
arrays of <byte>, <double-byte>, <machine-word>, <integer>,
<single-float>, and <double-float> and otherwise returns an
instantiable limited collection type.

where limited-stretchy-vector returns builtin class
representations for stretchy-vectors of <byte> and <byte-
character>, and otherwise returns an instantiable limited
collection type.

where limited-table always returns an instantiable limited
collection type.
```

7.5.6 Run-time limitations

- **`as`** does not check that its return value has the type of its first argument.
- Unicode is not implemented.
- The Dylan Reference Manual specifies that when comparisons are done between rational and floating point numbers, the floating point numbers be converted to rational and an exact comparison performed. Harlequin Dylan converts the rationals to floating point. This can produce inaccurate results.
- The type **`<extended-float>`** is equivalent to **`<double-float>`**. This is not, strictly speaking, a limitation, but you should be aware of it.

7.5.7 Compiler limitations

There is no way to loosely bind between two libraries compiled in Production mode.

7.6 Development environment problems

- Definitions of libraries and modules may have name clashes between (respectively) modules and bindings they import from other libraries and modules. The Dylan language provides ways to resolve these clashes, by excluding, prefixing or arbitrarily renaming module and binding names. See the DRM, Chapter 14, for details. Harlequin Dylan detects the clashes when you try to build a project containing them, but not before.

  In particular, the New Project Wizard may generate code which creates such clashes. Harlequin Dylan automatically resolves them by arbitrarily choosing one of the alternatives. Warnings appear each time you build the project until you manually resolve any clashes. You may want to resolve clashes yourself because the chosen alternative may not be the one you want.

7.7 C-FFI library

In the C-FFI library, FFI type-defining forms cannot be compiled in Interactive Development mode. These are: **`define C-struct`**, **`define C-union`**, **`define C-subtype`**, **`define C-mapped-subtype`**, and **`define C-pointer-type`**. Attempting to do so could cause internal errors in the compiler.

Other FFI defining forms, such as **`define C-function`** and **`define C-callable-wrapper`** should work in Interactive Development mode, but it is recommended for this release that, as far as possible, FFI definitions be separated out into a Production mode library for use in client libraries that are then free to exploit incremental compilation.

7.8 Non-local exits

Non-local exits from stack frames between Dylan and foreign code should be avoided whenever possible. A non-local exit occurs when control is transferred out of a region of code before reaching the end of the region, and then, if the region is a function body, returning from the function. Non-local function exits across the boundary between Dylan and foreign code are not fully supported. Non-local exits between foreign stack frames that pass through Dylan stack frames are not supported at all, and may cause out-of-language errors. Non-local exits between Dylan stack frames via foreign stack frames are partially supported. For more information on this, visit the Product Support page on our Web site.

7.9 OLE library macros

[Professional and Enterprise Editions]

- For a “dual” COM interface defined by the **`define dual-interface`** macro in the “ole-automation” library, the generated type info is incorrect. The Microsoft documentation says that the coclass refers to a TKIND_INTERFACE with the flag TYPEFLAG_FDUAL, which in turn is linked to a TKIND_DISPATCH. In actual practice, the coclass points to a TKIND_DISPATCH with TYPEFLAG_FDUAL which points to a TKIND_INTERFACE which also has TYPEFLAG_FDUAL and points back to the dispatch interface. (Bug 3914)

- Our type library support does not know how to record non-Automation data types properly.(Bug 3915)
- Names of members in custom and dual interfaces need to be distinct from members in other interfaces because of a name collision for the internal generated FFI wrapper functions. (Shows up as duplicate definitions for names **"CV_..."** and **"CCW_..."**.) (Bug 3917)
- If the **`define custom-interface`**, **`define dual-interface`** or **`define vtable-interface`** macros are used for defining a client only, the compiler will issue a serious warning of the form "Invalid type for argument this in call to method…" for each of the member functions. This results from the macro defining server-side stubs that call a generic function for which there are no applicable methods defined. These warnings can be ignored since they pertain to code that will not be executed anyway. You can suppress them by either defining dummy server methods or by adding a **`define open generic`** declaration for each of the functions. (Bug 4052)

7.10 SQL-ODBC library

[Professional and Enterprise Editions]

This section deals with problems relating to the SQL-ODBC library and to support for particular third-party databases.

7.10.1 General SQL-ODBC library problems

[Professional and Enterprise Editions]

- There is no support for large objects in the current version of the SQL-ODBC library.
- The SQL-ODBC library uses the Harlequin Dylan Date library. Currently, all date-time values retrieved from a database are managed as timestamps. Timestamp fields not present in the original date-time value use the default values provided by ODBC. The default coercion of these values is to convert them to instances of **`<date>`**. Fractional seconds are not currently supported.

7.10.2 Oracle ODBC support

[Professional and Enterprise Editions]

Oracle client software 7.3.2 has been tested with both Oracle ODBC drivers, and Intersolv ODBC driver version 3.01 and higher.

There is a known defect in Oracle 7.3.3 client software which prevents Oracle connections from being used under the Dylan environment with SQL-ODBC. This defect does not occur in Oracle 7.3.2. Oracle has a patch for this.

Verify that you require the patch. In the Oracle installer on the right hand column it should say what products are installed; check to see that the "required support files" version is at least 7.3.3.0.1a. Next, uninstall the Oracle ODBC drivers using the Oracle installer. Then, download the following file from the Oracle FTP server:

```
<URL:ftp://oracle-ftp.oracle.com/dev_tools/outgoing/ms*lang/
  odbc7/win32/n25310b.exe>
```

This is a self extracting executable; run it (in a folder hierarchy which does not contain any folders with names longer than eight characters) to extract 1381 files. Navigate down through the **`Win32\install`** folder; run the Oracle installer (**`orainst.exe`**). Upgrade the Oracle installer itself if necessary by clicking on the Oracle installer on the left-hand column, and choosing the **Install** button, and then install the ODBC drivers similarly. You may wish to have deleted your Oracle ODBC data sources first, in order to prevent the ODBC administrator from complaining that it cannot find the appropriate DLLs to uninstall these data source names. Then recreate your Oracle ODBC data sources.

We have verified SQL-ODBC with the Oracle 8 client software and Intersolv ODBC drivers version 3.11. Oracle7 client software was tested with Intersolv ODBC drivers version 3.01. Oracle's ODBC drivers do not appear to work in all cases, and thus we recommend using Intersolv drivers.

With Oracle backends, Dylan expects the value of the Oracle environment variable **`OPEN_CURSORS`** to be greater than 70. Unfortunately, in many Oracle 8 installations, the default value is 50, and thus must be changed on the server. This restriction will be removed in a future version of Harlequin Dylan.

7.11 Harlequin Dylan's CORBA® features

[Enterprise Edition]

This section covers issues with the Harlequin Dylan's CORBA® features.

7.11.1 Changes to CORBA features since Enterprise 1.1 beta

This section lists changes to Harlequin Dylan's CORBA features.

The runtime library (Harlequin-Dylan-ORB) no longer exports a lot of miscellaneous bindings that it uses internally, such as from the Format library and from the Harlequin-Dylan library itself. This was unnecessary and may have surprised users who defined clashing identifiers. It also affected the New Project wizard, which created CORBA projects that used both Dylan-ORB and Harlequin-Dylan. (Thus any applications that relied on these unnecessary exports must now use Harlequin-Dylan and / or Format explicitly.)

When forcing an in-process server to use the network and do marshalling, there is no longer a race condition during application cleanup. There should be no more socket errors in this case.

Context passing via the static invocation interface (stubs and skeletons) is now implemented. That is, an extra keyword argument is added to the signature as defined in the IDL to Dylan mapping.

The **`#pragma`** statements for controlling interface repository ID generation (ID, prefix, and version) are now interpreted by the IDL compiler.

The following server-side bindings are now implemented:

```
portableserver/<dynamic-servant>

portableserver/servant/primary-interface
```

NIL object references are now implemented.

7.11.2 STRUCT member overloading

[Enterprise Edition]

The mapping from IDL to Dylan for struct members is currently clumsy. Multiple structs or exceptions that define the same member name lead to multiple generic functions in the Dylan mapping.

7.12 ORB™ runtime

[Enterprise Edition]

This section covers issues with the ORB™ runtime.

7.12.1 ORB initialization limitations

The ORB initialization function **`corba/orb-init`** should only be called during initialization of an application's top-level Dylan library or afterwards during application execution. It should not be called during the initialization of a non-top-level Dylan library. The reason is that non-top-level Dylan library initialization currently occurs within the context of Windows DLL initialization which prohibits certain operations (for example, dynamically loading another DLL). Unfortunately, the initialization of Winsock2, which occurs during ORB initialization, violates the prohibition.

7.12.2 Port assignment limitations

[Enterprise Edition]

There is currently not much support for port assignment. The port number for the server must currently be set at the command line via **`-ORBport`** or via a setter method on the ORB, for example

```
orb-service-port(orb) := 9000;
```

In particular, there is no support for dynamic port assignment from the operating system. Nor is there any special support for creating IORs with an activator's port number. COMMENT: This is true in 1.2 but has already been implemented for 2.0

7.12.3 Not yet implemented

[Enterprise Edition]

The following constant, types, and functions are not yet implemented:

```
corba/$security
corba/<fixed>
corba/<long-double>
corba/<long-long>
```

```
corba/<servicedetail>
corba/<servicedetailtype>
corba/<serviceinformation>
corba/<serviceoption>
corba/<servicetype>
corba/<unsigned-long-long>
corba/<wchar>
corba/<wstring>
corba/context/delete
corba/fixed/digits
corba/fixed/scale
corba/nvlist/add-item
corba/nvlist/free
corba/nvlist/free-memory
corba/nvlist/get-count
corba/orb/create-list
corba/orb/create-operation-list
corba/orb/get-next-response
corba/orb/get-service-information
corba/orb/poll-next-response
corba/orb/send-multiple-requests
portableserver/<idassignmentpolicy>
portableserver/<idassignmentpolicyvalue>
portableserver/<iduniquenesspolicy>
portableserver/<iduniquenesspolicyvalue>
portableserver/<implicitactivationpolicy>
portableserver/<implicitactivationpolicyvalue>
portableserver/<lifespanpolicy>
portableserver/<lifespanpolicyvalue>
portableserver/<requestprocessingpolicy>
portableserver/<requestprocessingpolicyvalue>
portableserver/<servantretentionpolicy>
portableserver/<servantretentionpolicyvalue>
portableserver/<threadpolicy>
portableserver/<threadpolicyvalue>
portableserver/idassignmentpolicy/value
portableserver/iduniquenesspolicy/value
portableserver/implicitactivationpolicy/value
portableserver/lifespanpolicy/value
portableserver/poamanager/deactivate
portableserver/requestprocessingpolicy/value
portableserver/servantretentionpolicy/value
portableserver/threadpolicy/value
```

The interface to the Interface Repository is not yet implemented, including **`corba/object/get-interface`**. However, the **`TypeCode`** interface is implemented.

The `Interceptor` interface for hooking into request and message operations is not yet implemented.

The following GIOP / IIOP™ message formats are not yet fully implemented: COMMENT: Though they are for 2.0

```
cancelrequest
locaterequest
locatereply
closeconnection
messageerror
```

The `DynAny` interface is not yet implemented. This is used for traversing data structures embedded inside `Any`s when not enough type information is known to the program to coerce the `Any`s to more specific types.

7.12.4 No error checking

[Enterprise Edition]

The invocation order of DSI operations like `CORBA/SERVERREQUEST/ARGUMENTS` and `corba/serverrequest/ctx` are currently not checked.

There is currently no checking on the length of bounded sequences during marshalling.

There is currently no error checking if an inappropriate result is set using `corba/serverrequest/set-result` in the DSI.

There is currently no consistency checking between the typecode and the value when making an `Any` explicitly using the Dylan function `make`.

The IDL compiler does not yet check for clashing IDL identifiers and Dylan reserved words. Nor does it check for clashing attribute and operation names.

7.12.5 Incorrect API

[Enterprise Edition]

The arguments to `corba/serverequest/invoke` are currently the wrong way around.

The function `corba/orb/shutdown` currently ignores the value of the `wait-for-completion` argument.

The function **`corba/context/delete-values`** currently does not deal with trailing wildcards or unknown properties.

Currently the signature for **`portableserver/poa/create-poa`** not as implicitly specified in the IDL to Dylan mapping. Instead it is:

```
portableserver/poa/create-poa
  (poa :: portableserver/<poa>,
   adapter-name :: corba/<string>,
   poa-manager :: false-or(<poa-manager>),
   #key
   lifespan-policy
   id-uniqueness-policy
   id-assignment-policy
   implicit-activation-policy
   servant-retention-policy
   request-processing-policy
  => (poa :: portableserver/<poa>)
```

Moreover there is no means to affect the threading policy of the ORB which defaults to **`single_thread_model`**. That is, although the ORB uses multiple threads, there is only one thread per POA at present. If you want to control thread creation you can create new POAs.

The DII operations for specifying the list of context properties and the declared exceptions are not specified in the DII chapter of the CORBA 2.2 documentation. Instead **`ContextList`** and **`ExceptionList`** PIDL is defined in the C++ binding. However, currently signatures for functions performing these tasks are:

```
corba/request/add-context
  (request :: corba/<request>,
   property :: corba/<string>)
=> ()

corba/request/add-exception
  (request :: corba/<request>,
   exception-type :: corba/<typecode>)
=> ()
```

Neither the built-in APIs nor those generated from IDL support automatic coercion from Dylan values to **`Any`**s where the API expects an **`Any`**. The **`Any`** object must be made explicitly using **`as`** or **`make`**.

7.12.6 Miscellaneous

[Enterprise Edition]

The server will still try to send forwarding-info and exceptions to the client even in the **`oneway`** case.

Undeclared application exceptions will currently be returned to the client as **`corba/<bad-operation>`**. It is not clear if this is the correct exception to return.

Invoking requests on transient objects whose POA has since been destroyed and recreated are not presently trapped.

8 Supplemental Documentation

The documentation provided here supplements the Beta 2 documentation set.

8.1 Profiling

The Harlequin Dylan profiler takes periodic snapshots of the connected application, recording information about the progress of that application. Once this data has been collected, the information can be viewed in one of two ways. You can use the Profiler window, which presents the profiling results within a window where you can customize the presentation of the results. Alternatively, you can export the information as one of several different reports, primarily as ASCII text.

8.1.1 Profiling model

The profiler works by taking “snapshots” of a running application, and recording as much information as is available about the state of the application at that time. The following information is recorded for each application snapshot:

- the number of page faults since the last snapshot
- the “real” time spent since the last snapshot
- for each thread, every function on the stack, the amount of CPU time used by the thread since the last snapshot, and the amount of new allocation

There are three different profiling sampling styles:

- take a sample at a fixed rate

  This is the default sampling style, and is best used for understanding why a program is running slowly. The amount of CPU time spent in each thread is recorded.

- take a sample every time the application allocates

  This is a very intensive form of profiling as most applications allocate very frequently, but it gives extremely accurate results. In particular, for each snapshot, the class that was being allocated is recorded, which allows per class breakdowns of allocation patterns.

- take a sample every time the application hits a profiling breakpoint

  This feature is not implemented in Beta 2.

When profiling is switched off again, the results are collected and stored, ready to be viewed in the Profiler window, or to be exported.

8.1.2 Profiling user interface

When you want to start profiling, press the profiling button on the toolbar (normally in the Project or Profiler window). Any running program will then be profiled until you press the button again, at which point the results will be collected ready to be processed.

By default, the application takes samples at a rate of approximately one snapshot every 50 milliseconds. You can change the style of sampling used, and the rate using the **View>Profiling Options…** menu item from the Profiler window.

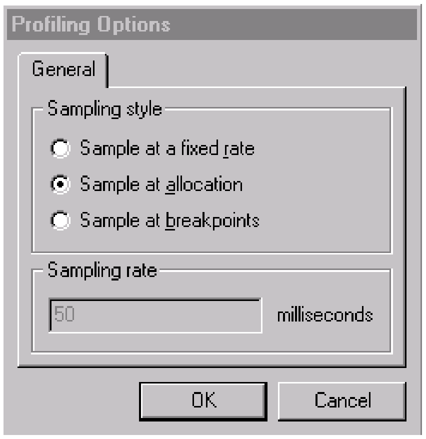
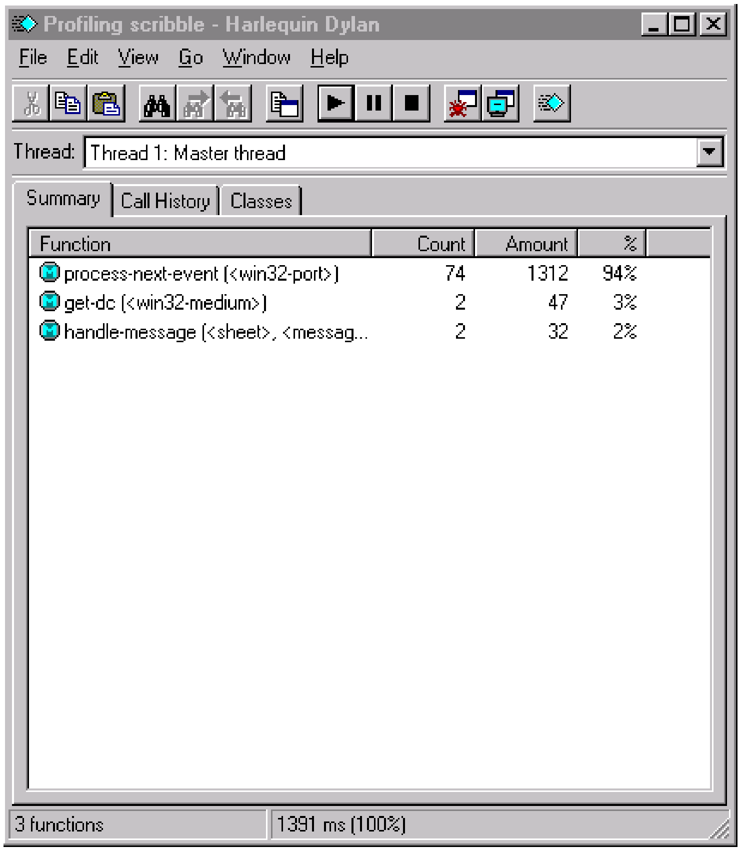


Figure 1 The Profiling Options dialog.

Once a profiling run has finished, you can use **View > Refresh** to make the Profiler process the results and present them to you. Here is a sample profiling run using interval profiling. Each function that is shown was found once or more at the top of the stack (the count column shows how many times). The amount column shows how long the particular function was at the top of the stack in total over the whole profiling session (the time is in milliseconds), and the **%** column shows the percentage breakdown of total CPU time (see Figure 2). Note that clicking on a column will cause the data to be sorted with respect to the values in that column.

As with all other tools in Harlequin Dylan, you can double-click on any function in the table to browse it, and you can right click to get a popup menu with possible actions.


Profiling scribble - Harlequin Dylan
File Edit View Go Window Help
Thread:
Thread 1: Master thread
Summary
Call History
Classes

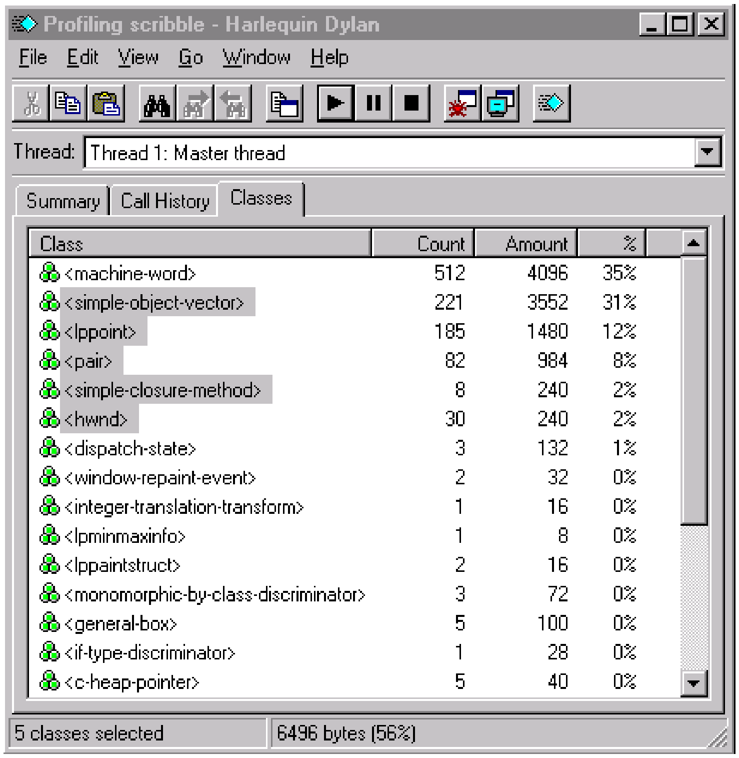
Function	Count	Amount	%
process-next-event (<win32-port>)	74	1312	94%
get-dc (<win32-medium>)	2	47	3%
handle-message (<sheet>, <messag...	2	32	2%

3 functions

1391 ms (100%)

Figure 2 The Summary tab page of the Profiling window.

After an allocation based profile, the Classes tab page shows the breakdown of allocation by class, along with a count showing the number of times that class was allocated (see Figure 3).


Profiling scribble - Harlequin Dylan
File Edit View Go Window Help
Thread: Thread 1: Master thread
Summary
Call History
Classes

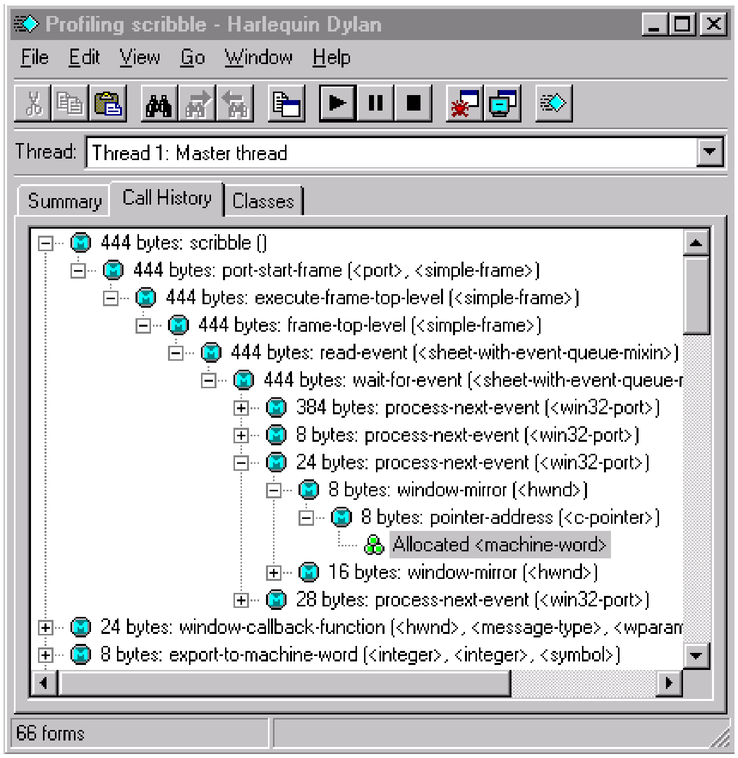
Class	Count	Amount	%
<machine-word>	512	4096	35%
<simple-object-vector>	221	3552	31%
<lppoint>	185	1480	12%
<pair>	82	984	8%
<simple-closure-method>	8	240	2%
<hwnd>	30	240	2%
<dispatch-state>	3	132	1%
<window-repaint-event>	2	32	0%
<integer-translation-transform>	1	16	0%
<lpminmaxinfo>	1	8	0%
<lppaintstruct>	2	16	0%
<monomorphic-by-class-discriminator>	3	72	0%
<general-box>	5	100	0%
<if-type-discriminator>	1	28	0%
<c-heap-pointer>	5	40	0%

5 classes selected | 6496 bytes (56%)

Figure 3 The Classes tab page of the Profiling window.

The Call History tab page works for both allocation and interval profiling (see Figure 4). This shows a breakdown of the state of the stack during the profiling period. The root nodes in the tree are the functions that are at the top of the stack, and expanding a node shows the calls that a function made at that time. When interval profiling, the time spent in the function is shown in milliseconds. When allocation profiling, it shows the allocation in bytes.

Note that using **Edit Source** from the popup menu on an allocated class (such as the highlighted item in the screenshot below) will take you to the line of code that caused this particular instance to be allocated (if the source for the library in question is available to you).


Profiling scribble - Harlequin Dylan
File Edit View Go Window Help
Thread: Thread 1: Master thread
Summary
Call History
Classes

```
444 bytes: scribble ()
  444 bytes: port-start-frame (<port>, <simple-frame>)
    444 bytes: execute-frame-top-level (<simple-frame>)
      444 bytes: frame-top-level (<simple-frame>)
        444 bytes: read-event (<sheet-with-event-queue-mixin>)
          444 bytes: wait-for-event (<sheet-with-event-queue-r
            384 bytes: process-next-event (<win32-port>)
            8 bytes: process-next-event (<win32-port>)
            24 bytes: process-next-event (<win32-port>)
              8 bytes: window-mirror (<hwnd>)
                8 bytes: pointer-address (<c-pointer>)
                  Allocated <machine-word>
              16 bytes: window-mirror (<hwnd>)
            28 bytes: process-next-event (<win32-port>)
24 bytes: window-callback-function (<hwnd>, <message-type>, <wparam
8 bytes: export-to-machine-word (<integer>, <integer>, <symbol>)
```

66 forms

Figure 4 The Call History tab page of the Profiling Window.

The standard **File > Export...** menu command can be used to export profiling results into other useful formats.

8.2 The Dylan Object Oriented Database

The Dylan Object Oriented Database (DOOD) is a simple mechanism for storing arbitrary objects and lazily loading them. During dump time, DOOD traverses a graph of objects and encodes the objects as a sequence of bytes. These bytes are later interpreted by DOOD during load time reconstructing an isomorphic object-graph in which cyclic structures and shared references are preserved. DOOD provides ways to control what slots in objects should be stored, and to decide what objects should be stored by proxies.

DOOD was meant to be a very Dylan simple object store that supported:

- pay as you go
- incremental loading,
- flexible proxies, and
- non-corrupting commits

DOOD was not meant to provide

- multiuser support,
- full blown transaction support,
- incremental writes,
- schema evolution, nor
- client/server support

In order to add persistence to a program, the easiest thing is to just save and load the data completely:

```
define method dump-data (data, locator)
    let dood = make(<dood>, locator: locator, direction: #"output",
if-exists: #"replace");
    dood-root(dood) := data;
    dood-commit(dood);
    dood-close(dood);
  end method;
define method load-data (locator) => (data)
    let dood = make(<dood>, locator: locator, direction: #"input");
    let data = dood-root(dood);
    dood-close(dood);
    data
  end method;
```

This works great for simple applications. More complicated applications potentially require support for data compression, special reinitialization, lazy loading of data, and multiple databases. Data can be compressed by being able to specify that only a subset of an object's slots are saved (for example, an object may cache information in a slot) or more generally by dumping the object's information in a completely different format on disk. The former technique is described in the schema section below and the latter technique is described in the proxy section below.

<dood> *Open primary class*

This class can be user subclassed and used as the basis for specialized loading and dumping behavior.

make *g.f. method*

`make(<dood>, #key name, locator, if-exists, stream, backups? = #t, version) => (result :: <dood>)`

creates and opens a dood over a file specified by the locator keyword parameter or over a stream given by the stream keyword parameter.

The **`locator:`** init-keyword should be a object valid for opening a stream with a **`locator:`** init-keyword.

The **`name:`** init-keyword is usually a string or symbol but user controlled.

The **`if-exists:`** init-keyword specifies the actions to take during creation of a file stream when the locator: init-keyword is specified. If **`if-exists:`** init-keyword specifies **`#"replace"`** then the database is considered empty.

The **`backups?:`** init-keyword specifies whether a new file is used during a commit. **`backups?`** is always false when the **`stream:`** init-keyword is specified.

The **`version:`** init-keyword specifies a version that can be used for version control. Upon opening of an existing database, the specified version is compared against stored version and if different a **`<dood-user-version-warning>`** condition is signaled.

<dood-warning> *sealed class*

superclass of all dood warnings.

<dood-corruption-warning> *sealed class*

signaled if DOOD data is found to be corrupted.

<dood-version-warning> *sealed class*

signaled if current DOOD version is different from saved DOOD version.

<dood-user-version-warning> *sealed class*

signaled if specified user version is different from saved user version.

dood-name *g.f. method*

```
dood-name (dood :: <dood>) => (name :: <object>)
```

returns the name of the specified dood.

dood-name-setter *g.f. method*

```
dood-name-setter (name, dood :: <dood>)
```

sets the name of the specified dood.

dood-root *g.f. method*

```
dood-root (dood :: <dood>) => (root :: <object>)
```

returns the one distinguished root of the specified dood. It is defaulted to be false when a new dood is created.

dood-root-setter *g.f. method*

```
dood-root-setter (root, dood :: <dood>)
```

sets the one distinguished root of the specified dood.

dood-commit *g.f. method*

`dood-commit (dood :: <dood>) => ()`

saves the data reachable from **`dood-root`**. When the **`backups?:`** init-keyword is specified to be **`true`**, the data is first written to a new file. The new file is named the same as the specified locator but with its suffix changed to “new”. Upon success, the original data file is replaced with the new file. Upon failure the new file is just removed and the original data file is untouched.

dood-size *g.f. method*

`dood-size (dood :: <dood>) => (res :: <integer>)`

returns the size in bytes of the data file.

dood-close *g.f. method*

`dood-close (dood :: <dood>, #key abort)`

closes the specified database and underlying file stream if created with l**`locator:`** init-keyword. If **`stream:`** init-keyword was specified then it is the user's responsibility to close this stream. Abort is passed through to close an underlying file stream.

8.2.1 Schemas

Schemas are declarative descriptions of how objects are persistently dumped and loaded. In the current version of DOOD, schemas are not themselves persistently stored. User versions can be used to manually ensure compatible schemas.

dood-class-definer *macro*

```
define dood-class class (superclasses)

    slot ...;
    lazy slot ...;

    disk slot ...;

    weak slot ... [ reinit-expression: ?:expression ] ...;

end dood-class;
```

The **dood-class-definer** macro defines a dylan class with extra slot adjectives specifying the dumping and loading behavior of the corresponding slot. The default DOOD treatment of a slot, called **deep,** is that its contents is recursively dumped and eagerly loaded. There are three dood slot adjectives that modify this behavior: **lazy, disk,** and **weak.** A **lazy** slot's contents is recursively dumped and lazily loaded, that is, loaded from disk upon first access. A **disk** slot's contents is recursively dumped and is always loaded from disk when and only when explicitly accessed and is never written back to the slot. A **weak** slot's contents is never dumped and a user can specify a **reinit-expression** to be used instead during loading. A **reinit-expression** must be specified even if an **init-expression** is the same, otherwise reinitialization will not occur and the slot will be unbound. In the current version of DOOD, the **reinit-expression** must appear as the first slot keyword parameter if at all. Accessing **lazy** slot values in a closed database will signal a **dood-proxy-error** (see below).

Example:

```
define dood-class <computation> (<object>)
    lazy slot computation-source-location :: false-or(<source-
location>) = #f,
      init-keyword: source-location:;
    slot computation-previous :: <compution>,
      required-init-keyword: previous:;
    slot computation-next :: <computation>,
      required-init-keyword: previous:;
    weak slot computation-type :: false-or(<type-estimate>) = #f,
      reinit-expression: #f;
  end dood-class;
```

8.2.2 Reading

Internally DOOD loads objects by instantiation and slot assignment. An object is instantiated via allocate, which returns an uninitialized instance, and then initialized by applying the setters of an objects class.

dood-reinitialize *open generic*

`dood-reinitialize (dood, object)`

For some objects the simple instantiation and slot assignment approach will not produce a well-formed object. **`dood-initialize`** gives objects a chance to correct any reconstruction problems. This function is called on an object immediately after the object has been loaded from disk.

Example:

```
define dood-class <rectangle> (<object>)
    slot rectangle-height :: <integer>,
      required-init-keyword: height:;
    slot rectangle-width :: <integer>,
      required-init-keyword: width:;
    weak rectangle-area :: <integer>;
  end dood-class;

define method dood-reinitialize (dood :: <dood>, object ::<rectangle>)
    next-method();
    rectangle-area(object)
      := rectange-height(object) * rectange-width(object);
  end method;
```

8.2.3 Tables

<dood-lazy-symbol-table> *concrete class*

provide a mechanism for indexes. The keys are symbols and are loaded lazily using a binary search. This is known to be an inferior layout strategy and will be replaced by b*-tree's in the future.

dood-lazy-forward-iteration-protocol *g.f. method*

`dood-lazy-forward-iteration-protocol (table)`

used for walking only keys presently loaded. The standard forward-iteration-protocol will load all keys and values into memory.

8.2.4 Proxies

Sometimes users need more control over how objects are dumped to disk. DOOD provide a general mechanism called a proxy, which provides both a disk representation of an object and a reconstruction policy. The basic idea is that during the dumping process each memory object is given a chance to provide a disk object (a proxy) to be used for dumping and then upon loading, a loaded disk object is given a chance to map back to its original memory object. Proxies can be used for mapping objects back to unique runtime objects, for compressing objects, for looking up objects in external databases, etc.

<do od-proxy> *open abstract class*

This is the superclass of all proxy objects. Users must subclass this class in order to define a new kind of proxy.

dood-disk-object *open generic*

`dood-disk-object (dood :: <dood>, memory-object) => (disk-object)`

Users write methods on this generic when they want an object to have a proxy. It returns a disk-object which is dumped in lieu of the memory-object.

dood-restore-proxy *open generic*

`dood-restore-proxy (dood :: <dood>, proxy :: <dood-proxy>) => (memory-object)`

This function is called immediately after a proxy is reconstructed with instantiation and slot assignment. Its job is to map from a disk-object back to its memory-object.

<dood-proxy-error> *sealed class*

signaled when proxy is restored from closed database.

8.2.5 Proxy examples

1. Dump by Reference

The first example shows how proxies can be used to dump objects by reference back to objects in a user's program. This is necessary when the data can not be dumped by DOOD (e.g., functions).

```
define constant $boot-objects = make(<table>);

  define class <boot-object> (<object>)
    slot boot-id :: <integer>, required-init-keyword: id:;
    slot boot-function :: <function>, required-init-keyword:
function:;
  end class;

  define method initialize (object :: <boot-object>, #key id, #all-
keys)
    next-method();
    $boot-objects[id] := object;
  end method;

  define class <boot-proxy> (<dood-proxy>)
    slot boot-id :: <integer>, required-init-keyword: id:;
  end class;

  define method dood-disk-object
      (dood :: <dood>, object :: <boot-object>) => (proxy :: <boot-
proxy>)
    make(<boot-proxy>, id: boot-id(object))
  end method;

  define method dood-restore-proxy
      (dood :: <dood>, proxy :: <boot-proxy>) => (object :: <boot-
object>)
    $boot-objects[boot-id(proxy)]
  end method;
```

2. Compression

The second example shows how proxies can be used to compress object's disk representation.

```
define class <person> (<object>)
    slot person-gender :: one-of(#"male", #"female"),
      required-init-keyword: gender:;
    slot person-height :: <integer>,
      required-init-keyword: height:;
  end class;

  define constant <person-code> = <integer>;

  define method encode-person (person :: <person>) => (code :: <person-
code>)
    if (person-gender(person) == #"male") 0 else 1 end
      + (person-height(person) * 2)
  end method;

  define method decode-person (code :: <person-code>) => (person ::
<person>)
    person-gender(person) := if (even?(code)) #"male" else #"female";
    person-height(person) := truncate/(code, 2);
  end method;

  define class <person-proxy> (<dood-proxy>)
    slot person-code :: <person-code>, required-init-keyword: code:;
  end class;

  define method dood-disk-object
      (dood :: <dood>, object :: <person>) => (proxy :: <person-proxy>)
    make(<person-proxy>, code: encode-person(object))
  end method;

  define method dood-restore-proxy
      (dood :: <dood>, proxy :: <person-proxy>) => (object :: <person>)
    make(<person>, decode-person(person-code(object)))
  end method;
```

3. Multiple Databases

The third example demonstrates how to use proxies for interdatabase references. Suppose that each database is registered in a symbol-table of databases and that each object stored in these databases knows both its name and to which database it belongs. Furthermore, suppose that each database has a lazy symbol-table stored as its distinguished root.

```
define class <dooded-object> (<object>)
    slot object-dood-name,    required-init-keyword: dood-name:;
    slot object-binding-name, required-init-keyword: binding-name:;
    // ...
  end class;

  define class <dood-cross-binding-proxy> (<dood-proxy>)
    slot proxy-dood-name,    required-init-keyword: dood-name:;
    slot proxy-binding-name, required-init-keyword: binding-name:;
  end class;

  define method dood-external-object (dood :: <dood>, name :: <symbol>)
    let symbol-table = dood-root(dood);
    element(symbol-table, name, default: #f)
  end method;

  define constant $doods = make(<table>);

  define method lookup-dood (name :: <symbol>) => (dood :: <dood>)
    element($doods, name, default: #f)
      | (element($doods, name)
     := make(<dood>, locator: as(<string>, name), direction:
#"input"))
  end method;

  define method dood-restore-proxy
      (dood :: <dood>, proxy :: <dood-cross-binding-proxy>) => (object)
    let external-dood = lookup-dood(proxy-dood-name(proxy);
    dood-external-object(external-dood, proxy-binding-name(proxy))
  end method;

  define method dood-disk-object
      (dood :: <dood>, object :: <dooded-object>) => (disk-object)
    if (dood-name(dood) == object-dood-name(object)) // local?
      object
    else
      make(<dood-cross-binding-proxy>,
     dood-name:    dood-name(dood),
     binding-name: object-binding-name(object))
    end if;
  end method;
```