Harlequin Dylan

Release and Installation Notes

Version 1.0 Beta 2

Europe:

Harlequin Limited
Barrington Hall
Barrington
Cambridge CB2 5RG
U.K.

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

North America:

Harlequin Incorporated
One Cambridge Center
Cambridge, MA 02142
U.S.A.

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

Electronic Access:

http://www.harlequin.co.uk/
http://www.harlequin.com/

Contents

Harlequin Dylan Release and Installation Notes

1 Introduction

Harlequin Dylan brings a new perspective to bear upon some of the problems routinely faced by all systems developers today. Dylan is object- oriented and combines the best facilities of late-binding languages like SmallTalk and static languages like C and C++.

Dylan's language design goals are simple:

- Promote modular, reusable, component-oriented programs
- Support powerful and familiar programming concepts
- Encourage rapid and productive development of programs
- Permit delivery of safe, fast, efficient, compact applications

This Beta release of Harlequin Dylan is intended to give you a taste of the product and an opportunity to try the advanced programming facilities that will be available with the full Harlequin Dylan release in early 1998.

Our purposes in having this Beta release are primarily

- To widen the user base for bug reporting to find bugs so as to improve the product.

- To provide data for market analysis in order to identify market segments.

We encourage you to try Harlequin Dylan out and report any bugs, so that we can address them for the full shipping release in early 1998.

There are two editions of Harlequin Dylan, the **Personal** edition and the **Professional** edition.

Personal edition: Consists of a robust, incremental, multi-threaded compiler, together with a powerful, dynamic development environment and an assortment of supporting libraries. An integrated editor, debugger and linker facilitate the delivery of EXE and DLL format executables, while a sophisticated GUI framework encapsulates the graphical Windows API within a Dylan context.

Professional edition: Extends the Personal Edition with the addition of a large layer of interoperability technologies including OLE / ActiveX support, ODBC connectivity to relational databases, and extensions to the GUI framework in support of OLE and ActiveX.

2 Installing Dylan

2.1 System Requirements

Harlequin Dylan runs on Windows 95 and NT 4.0 (Intel Pentium, Pentium Pro, Pentium II, or Pentium MMX). We recommend that you have:

- Internet Explorer 3.02 or later installed. See Section 2.5 on page 4.
- 64 MB of RAM or more.
- 150 MB of free disk storage for the product. If you are downloading the Personal from the Web, you will need an additional 70 MB during the installation process.
- A minimum of 128 MB of virtual memory, 160 or more preferred.

2.2 The linker

The GNU linker is supplied with Harlequin Dylan, and is the default. If you need to link up FFI libraries from C or C++ you may want to use the Microsoft Visual C++ linker version 4.2 or 5.0.

To choose a linker, you need to do two things. First, ensure the environment has the Visual C++ linker selected. Do this by Selecting **Options -> Environment Options...** in the main environment window. At the next dialog, select the Linker tab and then choose Microsoft linker.

Then make certain the environment variables are correctly set. The script **`VCVARS32.BAT`** is included with Visual C++ and does just this. You can either edit your own exec files to run this batch file on startup, or inspect the file and then set the environment variables by hand in the Environment tab of NT's control panel.

2.3 Distribution Format

Harlequin Dylan is distributed on CDROM and by FTP from Harlequin's World Wide Web site.

2.3.1 Installing Harlequin Dylan from the CDROM

To install the Harlequin Dylan from the CDROM,

1. Insert the CDROM in the CDROM drive.

   The image is auto-starting and brings up a menu. You may choose to read these Release and Installation Notes prior to doing the install.

2. Click on **Install**.

3. Select the version (Personal or Professional) you wish to install.

2.3.2 Retrieving Harlequin Dylan from the Web

The Personal edition is also available for downloading from
`http://www.harlequin.com/support/dylan/dylan-beta/dylan-beta.html`

This is a password protected location, and you will be sent or have received a password. From this location, you will be able to access the download site.

When the downloading dialog appears, you may select **Open** to install Harlequin Dylan immediately, or **Save to disk** to copy the files for later installation.

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

2.3.3 Licenses

The installer asks for a licensee name and license key and does not proceed with installation until a valid key is provided.

A license is valid only for a single edition of Harlequin Dylan. This means that the license keys for the same licensee are different for the Personal and Professional editions.

All beta test licenses expire on July 31, 1998, after which the installer refuses to install the product.

If your license data should become corrupted, the only way to re-register is to reinstall Harlequin Dylan.

2.4 Reinstalling Harlequin Dylan

To install later releases of Harlequin Dylan, you must first remove any existing copies of Harlequin Dylan. From the **Start** menu, select **Control Panels** under **Setting**s. Double click on **Add/Remove Programs,** select **Harlequin Dylan**, and click **Add/Remove**. This process does not remove any of your project files which are normally stored in the Projects folder. If files or folders are not removed, you can click on the **Details** button to see a list of those files.

2.5 Installing Internet Explorer to read online documentation

To use Harlequin Dylan's on-line documentation, you need Microsoft Internet Explorer 3.02 or later. You can download this from:

```
http://www.microsoft.com/msdownload/ieplatform/iewin95/iewin95.asp
```

Microsoft Internet Explorer 3.02 may need to be installed with some administrator privileges or it is possible that the Microsoft InfoTech Protocol extension, which HTML Help requires, will fail to register properly. We are still investigating this.

If in doubt, please check that the following registry entry exists using, for example, the program **regedit**:

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

If this registry entry does *not* exist it is an indication that the InfoTech protocol has not installed properly. In this case, try reinstalling Microsoft Internet Explorer 3.02 with administrator privileges. Alternatively, try reinstalling (with administrator privileges again) the specific ITSS.DLL using the program **regsvr32**:

```
regsvr32 itss.dll
```

You may also need to add the registry entry mentioned above explicitly.

3 Reporting bugs and contacting Harlequin support

3.1 Generating a bug report

The application **batch-debug** provided with Harlequin Dylan is useful for producing bug reports. If you are running in batch-mode, replace your usual call to dylan-compiler as follows:

dylan-compile ***options projects***

batch-debug dylan-compile -debug ***options projects*** **> compiler.log**

To run the Harlequin Dylan environment under the batch debugger, start it up as follows:

```
batch-debug harlequin-dylan > env.log
```

When running the environment under the batch debugger provided in the distribution, select **Cancel** in the error dialog to cause the batch debugger to emit a bug report when an error is encountered.

We are looking forward to your feedback on this beta release. We prefer communication by electronic mail, particularly when you submit bug reports.

3.2 Contacting Harlequin Support

You can contact Harlequin Support via e-mail at:

dylan-support@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:

Dylan Version	The version of the Harlequin Dylan you are using.
Machine & OS	The computer and OS version.
Description	A full description of the bug or question.
Repeat by	If a bug, step-by-step action to reproduce it.
Backtrace	Add a backtrace here, if you have one.

If you encounter a problem, a bug report with a full backtrace and small test case (if possible) is always appreciated.

4 Release notes

This section contains exceptions from documented behavior that are present in this version of the software. In some cases these represent bugs that will be fixed in the future and in other cases they represent last minute adjustments that did not make it into the documentation.

4.1 Known compiler problems

4.1.1 Forward References

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

There are no checks on unbound variables or constants during initialization. Thus forward references to unbound variables or constants can cause runtime crashes.

4.1.2 Potential interactive redefinition crashes

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

- Add new definitions. If you add a new sealed domain, however, your application will fail at runtime.
- Redefine methods
- Redefine a constant or variable to have a new value

Problems:

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

- Redefinition that change the type of an object, for example changing a class to a function or a constant to a variable.
- Redefinition of classes.
- Redefinitions from variable to thread variable.
- Redefinitions of module / libraries.

4.1.3 Incremental redefinition limitations

Redefinition of modules or libraries does not properly force all needed recompilation. If you are having trouble with compilation in the environment, you may want to try the stand-alone compiler, see Section 5.1 on page 14.

4.1.4 Run-time limitations

- When **`make`** creates a multidimensional array, it ignores the **`fill:`** parameter.
- Unicode is not implemented.

- Comparisons in numeric computations where numbers have been converted from or to floating point are not guaranteed accurate.
- The type **<extended-float>** is equivalent to **<double-float>**. This is not strictly speaking a limitation, but you should be aware of it.
- The Big-Integers library does not provide arbitrary precision integers. Only 64-bit **<integer>**s are supported. (This is unlikely to change in 1.0.)
- Multiplication does not always signal overflow when using Big-Integers.
- Some forms of division are not yet implemented when using Integers. Specifically, dividing one integer by another does not work. Use the **truncate** family of functions if you need integer division.

4.1.5 Compiler limitations

- The following are the permitted type expressions in tight mode:

  Names of things defined via define class and define constant.

  Literal constants.

  Expressions involving the functions: **type-union**, **singleton**, **false-or**, **one-of**, and **subclass**.

  limited, where the first argument is **<integer>**, or a subclass of **<collection>**. (**limited(<collection>, ..) => <collection>**, for now though.)

  Macro expansions (which expand into things that are evaluated at compile-time).
- **limited** on collection classes (that is, **limited(<array>, ...)**) is a stub returning the given class.
- There is no loose binding between two tightly compiled libraries.
- Constant folding of expressions containing integers can result in loss of precision of the upper two bits.
- Currently it is not possible to interrupt a compilation (for example, by pressing **Crtl-c**) without exiting the compiler.

- Full 8-bit characters are not yet supported in source code, although they are supported by the Streams library.

4.1.6 Parsing

Top level forms are currently required to have a terminating semicolon.

The *Dylan Reference Manual* (DRM) allows the semicolon after the last top level form of a source record to be omitted, but doing so in Harlequin Dylan will result in an "unexpected end of input" warning from the compiler, and the last form is skipped.

4.2 Known environment problems

4.2.1 The environment

- Window titles may differ slightly from the titles given in the documentation.
- There is no **Deselect All** command. Click somewhere else in the window to deselect objects.
- The **View > Options** command displays "Not yet implemented" for unimplemented Project or Browser features.
- **Go > Browse Project** requires a Dylan project database to operate. If there is no database available, it prompts you to parse and build a database.
- Progress reports for parsing are per library. They are interleaved.
- Windows sometimes open behind other windows. If nothing seems to have happened when you expect a window to open, check the Windows Task Bar to see if the window is in fact open. Some dialogs (for example, the dialog for opening a file) do not have buttons in the Task Bar. In these cases, try clicking in the window where you selected the command; this may bring the dialog to the front.

4.2.2 The editor

The editor has two keyboard layout modes, Windows and Emacs. The default keyboard layout is Windows. Use of Emacs key commands in Windows layout may yield unexpected results.

To switch file buffers in an editor window, select the **Go > File...** menu. If you're in "Associate each opened file with a separate window" mode, this reuses whichever window is currently showing that buffer, and bring it to the front, or open a new window if needed. If not, switching file buffers will reuse the current window, so acts like **C-x b** in Emacs.

- Tool bar:

  If you click on a toolbar button, the button may steal the focus. The workaround is to click on a non-Editor window, then click back on the Editor window.

- Status Bar

  The status bar has the following four fields, from left to right:

  Message area. Various messages appear here. To clear this area, use **View > Refresh** (shortcut **F5**).

  Mode indicator. Different file types may have different modes, in which new commands are available, or commands change their function. Three main modes are used: **`dylan`** (for Dylan source files), **`text`** (for text files) and **`fundamental`** (for files of unknown type).

  Document Modified? indicator. This is blank for unmodified documents, but changes to show **`Mod`** when the document is edited. Saving the document resets it to blank.

  Read-only? indicator. This shows **`R/W`** for editable documents, or **`R-O`** for read-only files.

- Items on the **Object** menu act on the selection, or on the word nearest to the cursor, if there is no selection.

- Features

  The editor menus do not show shortcuts, but they are available nevertheless. In fact, some shortcut keys may work even when menu items (or toolbar buttons) are disabled.

4.2.3 The Harlequin Dylan GUI debugger

Note: Unhandled errors bring up error dialogs that offer the option of debugging the environment. There are three buttons: **Yes**, **No**, and **Cancel**.

The workaround button is **No**, which puts you back to a point before the error occurred. Some errors are recursive, and in those cases **No** does not work.

Cancel invokes the system's installed debugger on Harlequin Dylan, and **Yes** quits the Dylan environment (thus exiting everything).

If your system does not have an installed debugger, a memory fault may occur. You should not choose **Cancel** unless you have a debugger (such as **WNDBG**) installed on your system.

Things to notice about the Harlequin Dylan debugger:

- The **Edit** menu operates only on the interactor pane, no matter which pane has the focus.
- The **Find-**related menu items in the Debugger are not working.
- When you are in the debugger, you should be aware that some function call frames may be optimized away by the compiler by, for example, inlining or tail-call optimization.
- You have to resize the debugger window manually in order for the component windows to sort themselves out properly after selecting one of the **View > Maximize ...** options.
- In Windows95, **exe**s and **dll**s linked with Visual C++ version 5 become locked by the debugger such that it is impossible to re-build them. You must reboot to remove the lock. (Workaround: we provide the GNU linker, which can be used instead).
- The following menu items on the Debugger's **Go** menu do not work:

  Goto Top of Stack

  Move Up Stack

  Move Down Stack

  Goto Bottom of Stack

4.2.4 Running an application within the environment

The *Environment Guide* describes the **Application > Start** and **Application > Debug** commands. **Start** goes right on through and enters the application while Debug pauses just before a named application entry point. However, there is currently no user interface for setting the entry point function. Instead the application is automatically paused as it starts to initialize the Dylan code associated with the topmost library. (If it were to be allowed to complete the initialization, then the application would call its entry point and the opportunity for initial debugging and interactive execution would be lost.)

4.2.5 Adding files to projects

- If you add a non Dylan file to a project, you get a warning:

  `"This file type not recognized (yet)."`

  The work around is close the project, edit the **`.hdp`** file by hand, open the project again.

- Attempting to insert a file from a different directory into a project does not work. The file is inserted, but it cannot be found when you close and reopen the project file.

- When you insert a new file into a project, the error **`"The file is already in the project and cannot be added again."`** is provoked. The file is correctly inserted into the project, however.

4.2.6 Project search

In this release the user projects are not searched for automatically. You have to open them explicitly, or you are asked to locate them if another user project uses them.

4.2.7 Project window errors tab

Errors or warnings not associated with a definition are displayed as children of the library in which they occur.

4.3 Known problems with interoperability

4.3.1 Foreign function interface

FFI type defining forms cannot be compiled in incremental 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 may result in internal compiler errors.

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

4.3.2 SQL-ODBC library

If your version of Harlequin Dylan has the OLE and ODBC extensions, please note the following:

- There is no support for Large Objects in this beta version of the SQL-ODBC library.
- The SQL-ODBC library uses Harlequin Dylan's Date library. Currently, all date-time values retrieved from a database are managed as timestamps. Timestamp fields not present in the orignal 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.

4.4 Other known bugs

- Compiler warnings of the form

  `The binding` ***`foo`*** **`is defined but not referenced or exported.`**

  are written to the **`.log`** file, but do not appear (and are not counted) in the environment's **Errors** display.

- The line **`Compilation-Mode: incremental`** in a **`.lid`** file vanishes when the file is rewritten to include the **`major-version`** and **`minor-version`**

lines.

- Pausing and then resuming can freeze the environment. This is not a general problem and is caused by some not yet implemented functionality. Interactive evaluations are only supported at well-defined locations like breakpoints or Dylan errors, and in general can not be performed after pressing the **Pause** button at some arbitrary moment. As a consequence of this, the debugger offers to nudge the application to a point at which interactive evaluations are possible. However, this latter mechanism is not currently working.

5 Addendum to user documentation

5.1 The stand-alone compiler

The Harlequin Dylan compiler performs seven phases for each compilation:

Parsing

Computing data models

Computing code models

Performing type analysis

Optimizing

Dumping dispatch colors

Generating code

Harlequin Dylan provides two options for compiling Dylan code, the interactive development environment and the stand-alone compiler. The stand-alone compiler is primarily included as a separate application to support build scripting (such as **make** files), but it also provides a command line interface that some developers might find preferable.

5.1.1 Batch mode

The simplest mode of operation of the Dylan stand-alone compiler is *batch mode*, where it is invoked with some command line options and a list of projects. It performs the requested operations and then exits.

This mode is for doing one-off compiles, or for use in build scripts.

The projects can be specified either as a pathname to a Harlequin Dylan **.hdp** or **.lid** file, or as a library name in which case the compiler uses its standard library searching procedures to find the library's definition. See Section 5.8.2 on page 23.

If you are having trouble getting the compiler to accept a pathname, use the DOS version of the pathname. Some file system folders seem to cause problems, for example **C:\Program Files**.

The default operation in batch mode is to compile only the libraries that have changed in a project. So:

```
dylan-compile c:\dylan\my-project\my-project.hdp
```

incrementally compiles as needed. Similary:

```
dylan-compile my-first-library my-second-library
```

compiles the two specified libraries if it can find them (it exits with an error code if it cannot).

The following useful options can be specified as command line arguments to dylan-compile.

/help, **/?**	give some brief help on using dylan-compile
/logo	displays the Harlequin copyright banner (the default)
/nologo	suppresses the Harlequin copyright banner
/all	compile the library and all its dependents
/tight	make the project be in tight mode
/loose	make the project be in loose mode
/nocompile	do not do a compile
/save	save a compiler database
/link	link the project's object files

/nolink	do not link object files (the default)
/exe	link the project as an executable (the default)
/dll	link the project as a DLL
/microsoft	link the project using the Microsoft linker
/gnu	link the project using the GNU linker (the default)
/debug	enter a debugger if the compiler crashes

The **/debug** option is provided to simplify the process of reporting bugs. Instead of aborting the compilation in the face of an internal compiler error, it instead crashes which allows a debugger to get a backtrace. See Section 3 on page 5.

Some useful scenarios:

4. Compile and link a library as an executable

   ```
   dylan-compile /link my-executable
   ```

5. Compile and save a library, and link it with the GNU linker

   ```
   dylan-compile /save /gnu my-dll
   ```

6. Recompile a project from scratch and link it as an executable

   ```
   dylan-compile /all /exe c:/dylan/my-project.hdp
   ```

5.1.2 Console mode

The *console mode* for dylan-compile is provided to allow you to perform multiple compiles over a period of time without having to restart the compiler every time. To start the compiler up in console mode, just run it without any arguments.

When dylan-compile starts up in console mode, you are presented with the Harlequin copyright banner and then a command prompt. You can now issue commands to the compiler one at a time, and it performs them for you.

compile-changes

Usage: **`compile-changes`** *project*

The command compile-changes is equivalent to the environment's **Compile Changes** menu. It is also the default operation performed when running the compiler in batch mode. This operation performs an incremental compilation, that is it recompiles the specified project only enough to take into account any changes that have been made. Thus simple changes to one line of a file should be much faster than if almost every file in the project has changed.

compile-all

Usage: **`compile-all`** *project*

The command **`compile-all`** is equivalent to the environment's **Compile All** menu. It is the same as running the compiler in batch mode with the **`/all`** option. It recompiles the specified project and all used projects from scratch. This command is typically only needed when a full compilation needs to be guaranteed, such as when recompiling a project in order to release it.

link

Usage: **`link`** *options project*

The command **`link`** is equivalent to the environment's **Link** menu. It does no compilation, but just links whatever build products already exist, and stores them in the project's bin directory.

The available options are:

`/dll`	link the project as a DLL (default: executable)
`/force`	force link the project and its subprojects
`/not-recursive`	only link this project, not its subprojects
`/gnu`	link using the GNU linker (the default)
`/microsoft`	link using the Microsoft linker

The Microsoft linker generates its output to **`standard-output`**.

Note: The GNU linker currently does not clean up when it encounters an error. It leaves many temporary files which you must clean up manually before you attempt to link again.

build

Usage: **`build`** *options project*

This command is equivalent to using **`compile-changes`** and then **`link`** on the project. All of the options for **`compile-changes`** and **`link`** are accepted. **`build`** is equivalent to the environment's **Build** menu.

open

Usage: **`open`** *project*

Opens a project either by library name or filename. Once a project is opened, compilation of any libraries that use the library defined by the project use this opened definition.

parse

Usage: **`parse`** *project*

Parses a project, producing an in-memory model of the code. This lets you pick up syntax errors without going through a whole compile. The project can be specified either as a library name or as a filename.

import

Usage: **`import`** *project*

Creates a project file for a given Library Interchange Definition (**`.lid`**) file. The project can be specified either as a library name or as a filename.

close

Usage: **`close`** *project*

The **`close`** command removes all knowledge of a project from the Dylan compiler, so for example you can compile a different version of that project.

close-all

Usage: **`close-all`**

The **close-all** command closes all open projects. You probably want to do this when you have finished with one set of projects and want to start compiling a completely different set.

help

Usage: **help**

The help command with no argument displays all the commands available along with a brief description of how to use them. If you specify a command (**help close-all**, for example), it displays the documentation on how to use the command.

exit

Usage: **exit**

The **exit** command exits the compiler, and there is also an alias, **quit**, provided for convenience.

5.1.3 Compiler errors and warnings

There are three levels of warning or error that the compiler emits. Each has its own icon to distinguish it:

warning	Something is wrong but the application will build and probably will not crash at runtime, for example, a binding defined but not referenced or exported.
serious-warning	The application will build but may crash depending on actions at runtime, for example, a type error in a method that may or may not get called
error	The application probably will not build, probably will crash (maybe in DLL init), for example, a congruency error between a generic function and a method

5.2 User project locations and compiler products

When you create a project, you create the **.hdp** file in the same directory as the sources for the project application. When you compile the project, the compiler creates three sub-directories in your project directory. These are:

1. **`build`**

   This directory holds internal compiler build products, such as files of type **`.obj`**, **`.ddb`** (database files), and a few others.

2. **`bin`**

   This directory holds the executables (**`.exe`**) for your project as well as the files for shared libraries (**`.dll`**) used in your project, since these files must be in the local project directory to be used.

3. **`lib`**

   This directory holds the linker products, **`.lib`** files for the Microsoft linker and **`.defs`** files for the GNU linker.

The last two directories are necessary if you need to link another project against this one.

Contrary to a statement in the *Harlequin Dylan Environment Guide* user projects are not registered anywhere and no such feature is planned.

5.3 Adding duplicate files to projects

If you attempt to add a duplicate file to a project, a warning is displayed:

`"The file `*`filename`*` is already in the project and cannnot be added again."`

5.4 Project tool Application menu

The *Environment Guide* describes the project window **Application** menu items such as **Application > Build othello.exe**, **Application > Link othello.exe**, and **Application > Link othello.dll.** The intention is that in each project window, the file names in these menu items will be formed using the name of the current project as the file name stem.

In every project tool in the beta, these menu items are **Application > Build**, and **Application > Link**.

The **Application** menu (shared by all main top level windows) has a **Threads...** menu item (after **Interact** in same item group) that launches an object browser on the application object and makes **Threads** the selected tab.

The **Threads** tab shows a table of application threads (1st column) and their current states (2nd column).

Browsing a thread (via double clicking, popup menu, or **Object** menu, for example) stops the application and launches a debugger window looking at that thread.

The right mouse button menu items are also available on the objects in the stack pane: thread, stack frames, and local variables. Double clicking on an item in the stack pane does the same as **Browse** from the right mouse button menu.

If you pause an application and the pause point is not at a suitable interaction point the environment informs you and offers to nudge the application on to an interaction point. This may not work.

5.5 Threads

Dylan threads are real Win32 threads. They run concurrently. Blocking on I/O on one thread does not stop all your Dylan threads, you can still call-in to Dylan from C on different threads.

5.6 Debugger

The **Save Bug Report** dialog has an option, **Message Log**, which, by default, lets you append the current thread's message log to the rest of the backtrace dumped.

You should be aware that some internal stack frames, concerned with Generic Function dispatching, are hidden by default.

The Debugger has an option box in its toolbar for switching the current module, just like the Browser.

5.7 Operating system library

The operating-system **`run-application`** function has two new keywords to add more control over how the command's shell appears:

`minmize?:` If **`#t`**, the shell appears minimized.

`activate?:` If **`#t`**, the shell window becomes the active window.

There is an additional new keyword to control progress notes during builds:

`output-handle:` *file-handle of dylan output-stream*

the running application will redirect its output to that stream instead of using a console-stream.

5.8 GNU linker pathnames

The GNU Linking tools cannot use pathnames that have spaces in them as search paths for their standard libraries. The following function coerces pathnames into usable form for the GNU linker:

```
shorten-pathname (path :: <pathname>) => (shortened-path ::
<byte-string>)
```

5.8.1 Recovering from errors

abort

When the compiler has crashed, this command allows you to abort that operation so that you can continue without having to restart the compiler from scratch. If you'd like to report the problem, you should use the **`debug`** command while you are running the compiler inside a debugger (see the **`batch-debug`** application, Section 3.1 on page 5, that is provided to help you to generate bug reports).

continue

When the compiler has crashed, this operation allows you to choose between one of the available restarts by specifying the restart number. Typically you just want to abort the operation, for which the command **`abort`** is provided (use **`help`** for more details).

debug

When the compiler has crashed, this command allows you to enter a debugger so that you can determine what is happening. Generally speaking, you should use the **`batch-debug`** application as your debugger, as this produces a bug report that you can send to Harlequin. See Section 3 on page 5 for more details.

5.8.2 Searching

Harlequin Dylan's provides searching commands in a Find Window. The Find Window has two text fields and some checkboxes:

Find In this field you enter the text for which to search.

Replace In this field you enter the text with which to replace the matched text, if desired.

Both of these fields are combo boxes, and contain the ten most recent Find / Replace strings for quick access to previous strings.

Checkbox options:

Batch When selected, all matches are found and collected into a list that may be browsed, otherwise, search commands find and display one match at a time.

Wrap When selected, and searching reaches the end of data, searching automatically continues from the start of data. When multiple search targets are selected, this wraps from the last target to the first (when searching backwards, wrapping proceeds from the end of data or the last target).

Match Whole Words When selected, searching only matches whole words. For example, when selected, "this" will only match "this" and not "thistle". When searching UI items in a tree or list, for example, this option matches the entire item name, not just whole words inside the name.

Match Case When selected, searching is case-sensitive. For example, when selected, "this" will only match "this" and not "This" or "THIS".

Look In This option box is used to select "search targets" in which to search. Currently, the only available targets are windows, such as project and browser windows.

The first item is "the current window". The current window is set by certain events: When the **Find...** command is selected or when a window is opened When there is no current window, this item reads "(None)". Basically, this attempts to keep the "current window" set to the frontmost searchable window, so if the user clicks on a window to bring it to the front, then selects a search command in the Find window, searching will proceed in the clicked window.

The second item is **Others...**. It allows the user to select more than one search target.

When searching multiple targets and the end of target data is reached, the next target is searched from the start of data (when searching backwards, of course, searching proceeds from the end of data in the previous target).

Buttons:

Find Opens the **Find** window, which contains search options.

Replace Replaces the current selection with the **Replace** string, leaving the replacement text selected, but only if the selected text matches the search options, otherwise, it does nothing (actually, it beeps to provide feedback).

Replace & Find Does the equivalent of **Replace** followed by **Find Next**. If the current selection does not match, it is not replaced, but the next match is still searched for

Replace All Does the equivalent of **Replace** followed by **Find Next** until there are no more matches.

Stop The Stop button stops the current search / replace command.

The **Go Menu** contains searching commands:

Find... Opens the Find window, which contains search options.

Find Next Searches for the next match starting after the current selection (or insertion point).

Find Previous Searches for the previous match, starting before the current selection (or insertion point).

Find in Next File Searches for the next match in the next search target, starting from the start of the data instead of at the selection.

Find in Previous File Searches for the previous match, in the previous search target, searching from the end of the data, instead of at the selection.

Enter Find String Copies the selected text to the **Find** text in the Find window.

Enter Replace String Copies the selected text to the **Replace** text in the Find window.

Find Selection Does the equivalent of **Enter Find String** followed by **Find Next**.

Find Selection Previous Does the equivalent of **Enter Find String** followed by **Find Previous**.

Replace Replaces the current selection with the 'Replace' string, leaving the replacement text selected, but only if the selected text matches the search options, otherwise, it does nothing (actually, it beeps to provide feedback).

Replace & Find Next Does the equivalent of **Replace** followed by **Find Next**. If the current selection does not match, it is not replaced, but the next match is still searched for

Replace & Find Previous Does the equivalent of **Replace** followed by **Find Previous**. If the current selection does not match, it is not replaced, but the next match is still searched for.

Replace All Does the equivalent of **Replace** followed by **Find Next** until there are no more matches.

Note: The shortcut keys for these commands are defined such that for commands that have Next / Previous pairs, the Previous variant can be selected by using the Shift key. For example, **Find Next** is **F3**, and **Find Previous** is **Shift+F3**.

The Toolbar contains some searching command buttons:

Find... Opens the Find window, which contains search options.

Find Next Searches for the next match starting after the current selection (or insertion point).

Find Previous Searches for the previous match, starting before the current selection (or insertion point).