Harlequin Dylan Release and Installation Notes

Beta Preview

Contents

Harlequin Dylan Release and Installation Notes

1 Introduction

This 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 later this year.

Our purposes in having this Beta Preview release are several:

- To involve important customers early along in the product life cycle
- To get feedback on usability and user interface issues
- To gain practice with and expertise in the release process.

This release is a preview of the final Harlequin Dylan product. It is neither a complete nor an optimized release. Most notably, this release does not include OLE or ODBC, but they will be included in the full beta in the fall. There are many bugs, the size of the image is too large, and performance is slow. We will be working on all these factors between now and the full beta release later in the fall, with the goal to have a full shipping release by the end of the year.

2 Installing Dylan

Harlequin Dylan runs in Windows 95 and NT 4.0 (Intel 486 and Pentium). You need:

- 64 MB of RAM
- 150 MB of free disk storage for the product,plus an additional 70 MB used only during the installation process.
- A minimum of 128 MB of virtual memory, 160 or more preferred.
- MS Visual C++ linker, version 4.2 or higher.
- The PATH, LIB, and INCLUDE environment variables should be set correctly to find the VC++ linker and associated files.

The image is available for downloading from **`http://www.harlequin.com/~dylan....`**

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

To start Harlequin Dylan, choose **Start > Programs > Harlequin Dylan > 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.

3 Release notes

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

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

Cancel attempts to debug the main Dylan environment thread, and **Yes** quits the Dylan environment (thus exiting everything).

3.1 The compiler

The compiler can operate in either of two modes, *Minimize static dependencies* (or *loose* mode) and *Maximize speed* (or *tight* mode).

Tight mode is a traditional optimizing compiler.

Loose mode uses techniques to defer activity to the run-time instead of compile time. This has several effects:

- Compilation is faster because the compiler does less modeling of what is going to happen in the run-time, and is able to do fewer optimizations.
- Re-compilation is faster because the compiler introduces fewer inter-definition dependencies. Since there are fewer dependents on any given definition, fewer things have to be recompiled when a definition changes.
- Redefinition (at run-time) is easier to implement, and has better semantics. Since the shared inter-definition assumptions are minimized, a definition can be changed with less runtime impact than in tight mode.

There are, however, some drawbacks:

- The compiler detects fewer errors, since it does less modeling of what is going to happen at run-time. This can be useful for rapid prototyping and similar development projects.
- Code that runs correctly in tight mode does not always work in loose mode and vice versa. This is a limitation for this release which we hope to remove in the future. For example

  - sealing violations cause crashes in tight mode but are ignored in loose mode.

  - forward references in code do not matter in tight mode but cause crashes in loose mode.
- Loosely compiled code runs more slowly because of the fewer optimizations.

Note: Warnings generated while compiling code appear in a DOS console window, not in the warnings display of the environment as planned. In the future these will be browsable in the environment.

3.2 Compiler bugs and limitations

3.2.1 Potential run-time crashes

- No function type checks — calling or applying a non-function results in out of language crash.
- Return values type check disabled — compiler can make assumptions based on return values types which could make user code crash if it returns objects not of declared types.
- Crash on forward references — no checks on unbound variable/constants during initialization. This results in debugging headaches.
- Congruency checks are not enabled — adding a method taking keywords to a generic that does not or vice-versa can result in a run-time crash.
- **`as(<symbol>`**, *string*) does not copy string

3.2.2 Potential interactive redefinition crashes

What you can do without risk:

- Add new definitions
- Redefine methods
- Redefine constant/variable to have a new value

Problems:

- Redefinition of generic signatures to a non congruent definition can cause crashes
- Redefinition between classes, functions, constants, variables is dangerous
- Redefinition of classes is dangerous
- Redefinition between variable to thread variable is dangerous
- Redefinition of module/libraries is dangerous

3.2.3 Incremental redefinition limitations

- Redefinition of module/libraries does not properly force all needed recompilation. If you are having trouble with compilation in the environment, you may want to try the Standalone compiler, see Section 3.2.7 on page 10.

3.2.4 Compiler crashes

- Virtual slot definitions that do not have corresponding (implicit) generic definitions crash the compiler.
- Variable definition in different module to creation will result in a compiler crash.

3.2.5 Run-time limitations

- Make of multidimensional array ignores the **`fill:`** parameter.
- Slot-initialized? is incorrectly sealed
- Unicode not implemented.
- Transitivity of numeric computations.
- 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 Big-Integers.

3.2.6 Compiler limitations

- Sometimes extra **`let`**-bound variables default to random values, not **`false`**.

- 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 thing which are evaluated at compile-time
- Limited on collection classes is a stub returning the given class.
- The **`local`** *`name`* **`(...) ... end`** abbreviation does not work. Use the full form, **`local method`** *`name`* **`(...) ... end`** instead.
- Poor warnings for undefined variables — warns many times.
- Overconservative compile-time method keyword checks — does not compute applicable keywords correctly for generic-function methods.
- Overconservative multiple values return type warnings.
- 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

3.2.7 The standalone compiler

If you have run into problems using the compiler in the environment, but still want to get your application to run, you can use the standalone compiler.

To start the standalone compiler, choose the **Dylan Compiler** shortcut on the **Start** menu, **Start > Programs > Harlequin Dylan > Dylan Compiler**.

It supports the following commands:

a	Select an abort restart
build-locations	Display build locations

c	Select a continuation restart
close	Close a Dylan project
close-all	Close all open Dylan projects
compile	compile a Dylan library (without dependency tracking — this is dangerous) Options: force-compile, force-parse, harp Example: compile -force-parse test
d	Enter debugger
exit	Quit standalone compiler
find-library	Find location of a Dylan library
help	Display Help information
load	Load a Dylan library Example: load test
p	Print available restart options
registries	Display available registries
update-libraries	Compile a Dylan application (with dependency checking) Options: force Example: update-libraries -force test

3.3 The debugger

There is no pane for the display of source code in the debugger in this release.

3.4 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.

3.5 Parsing

Top level forms in files, and when entered at the interactor, 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 with that last form being skipped.

The short form of the "local" declaration for binding local methods, **`local foo (x)`**, is not currently supported. The long form **`local method foo (x)`** is required.

3.6 The editor

The editor (Deuce) 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.

- 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.

  As in other environment tools, the combo box at the left of the toolbar is a "quick search" gadget. Enter a string to search for, then press Enter. For more search options, or for Replace, use the keyboard or menu items. The "history" on the combo box doesn't work in any environment tool.

  The last button on the toolbar shows information on the location of the cursor and the character at that point.

- Status Bar

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

1. A message area, where various messages will appear. To clear this area, use **View > Refresh** (shortcut **F5**).

2. A 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).

3. A "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.

4. A "read-only?" indicator. This shows "R/W" for editable documents, or "R-O" for read-only files. (A graphical bug means that this field is partly obscured by the resize grip, so these indicators are currently indistinguishable. However, the message area will display an error message if you try to edit a read-only file.)

- Items on the **Object** menu act on the selection, or on the word nearest to the cursor, if there is no selection. The **Describe** operation on the **Object** menu opens a window which shows a brief description of the definition of the object. [This latter may well not work.]

- For newly created text files, the **Save** command is disabled, but **Save As** allows you to save the file under a file name you provide.

  The **Save All** command (which is also invoked automatically when the last editor window is closed) will offer to save all modified files you have open, both new and opened-from-disk. However, it will silently fail to save new files.

  Also, when **Save All** is invoked because the last window is closing, the **Cancel** button will avoid saving the files, but will not stop the window from closing.

- The **Alt** key does not act like it does in Emacs, even when you are using the Emacs command set.

- Bug notes and workarounds

If you find the editing area fails to respond when you type into it, or that typing seems to activate toolbar buttons, or the editor just "beeps", try clicking outside the window, then inside it again.

If you try to open a file without any file extension (e.g., "Readme" instead of "Readme.txt"), the environment will signal an error, and probably crash soon afterwards. The simple workaround is, of course, not to do this.

You can cut or copy from the editor, and paste into other applications, but you cannot paste into the editor from other applications. You can cut, copy and paste within the editor, however.

- 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. See

3.7 The environment

Windows frequently 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 (e.g., 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.

If you double click on a node in the tree control, it may not do what you expect, namely launch the application. First single click on the node to select it, then double click.

When you use the menu to open a project file (**.hdp** file), it may be opened in the editor (Deuce) as opposed to being launched. To launch the project, select the **.lid** file instead.

The **Parse**, **Compile** and **Build** status bar dialogs have a bug in which they do not refresh unless mouse movement occurs over them. Bear this in mind if one of these operations appears to have hung — it may be complete but the dialog does not reflect it.

3.8 Running an application within the environment

The environment guide describes the **Application > Start** and **Application > Debug** commands. In the Beta Preview, neither of these commands works quite as documented.

There is no user interface for setting the entry function breakpoint. In addition, for both commands, the environment breaks to debugger automatically, and even earlier than described for **Application > Debug.** The effect is that both **Application > Start** and **Application > Debug** appear to behave like **Application > Debug** is expected to, but neither is quite doing so.

As a workaround for running applications within the environment, choose **Application > Resume** to resume an execution started with **Application > Start**.

When you exit your last Dylan window, the environment does not automatically exit. You have to go into the Task Manager and explicitly kill it.

3.9 Projects

For this release the planned *Project Wizard* is not implemented. As a result the following menu buttons/commands do not operate:

- The radio button **Create new project** in the initial dialog.
- The entries **Insert File..., Move File Up,** and **Move File Down** in the **Project** menu.
- Entries in the **Edit** menu in the project tool.

It is necessary to construct and edit new projects by hand. This is a temporary measure and is not recommended as a general practice.

To create a new project, you need to construct a library interface definition **`.lid`** file and a project file, the latter is optional but recommended.

A full description of the format of **`.lid`** files can be found in the *Core Features and Mathematics* section in the reference documentation. For a simple project, however, the following should suffice.

Imagine you wish to create a project which will consist of three source files, called **`library.dylan`**, **`core.dylan`**, and **`support.dylan`**. The library associated with the project is to be called **opair**. For simplicity, we assume that the library and the project share the same name.

A suitable **.lid** file might be named **opair.lid**, and would look something like this:

```
library:  opair
files:    library
          support
          core
```

Note that there *must not* be a blank line between the **library**: line and the **files:** line, and that the filenames appear in "bare" form, with neither directory nor extension present.

The corresponding project file should be called **opair.hdp**. It consists of two lines:

```
library: opair
lid: opair.lid
```

Changes to a project (e.g. adding or removing source files, or changing their order of initialisation) are accomplished by editing the **.lid** file.

3.9.1 Project tool application menu

The *Environment Guide* describes project tool 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 tool, 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 preview, these menu items will be **Application > Build xxx.exe**, and **Application > Link xxx.exe**.

From the project controller, **Application > Start** and **Debug** may signal an error, hang or crash the system if the project has not been compiled.

If you click on a project node to expand the definition before you have parsed or compiled the project, it renders that project unworkable. Do not click on a project node before you have compiled the project.

3.9.2 Project > Insert

The project tool's **Project > Insert** feature for inserting new source files into a project does not work in the beta preview. To insert a new source file into a (LID-based) project, add it to the LID file's `Files:` list. The source file name should appear on its own line and should not include the `.dylan` suffix.

3.10 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.

3.11 Documentation

The documentation available with this release is incomplete. It, like Harlequin Dylan itself, is still under development.

- The DUIM Reference Manual is only a skeleton, containing only function names and arguments (signatures).
- The DUIM Users Guide and the Environment Guide are incomplete.

More complete documentation will be available for the full Beta release scheduled for the fall.

4 Known environment deficiencies

- A number of menu items are not actually implemented. Most of these produce a "This is not implemented yet." notification, but others may still give UDE (Unhandled Dylan Error).

- When using the compiler and / or debugger, do not click on the Warnings tab before there is a project open or compiling. This can cause the software to crash
- There is a bug where the output from the running application goes into a window that disappears as soon as the application exits.

5 How to reach 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.

We are looking forward to your feedback on this preview release.

We prefer communication by electronic mail, particularly when you submit bug reports.

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

If you are within the environment and end up in the debugger, you can choose the **File > Save Bug Report** menu item. This generate a bug-report template you can fill in and send to us.

If you are not running the environment, you can use the **`:bug-form`** command in the tty debugger to generate a bug report.