Harlequin Dylan Release and Installation Notes

Version 1.0

Contents

Copyright and Trademarks

Harlequin Dylan: Release and Installation Notes

Version 1.0

June1998

Part number: DYL-1.0-RIN

Europe:

Harlequin plc
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

Welcome to Harlequin Dylan, and thank you for your interest in this exciting new development tool.

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

The release notes consist of:

- This product overview.
- Installation instructions
- Information on reporting bugs and accessing customer support.
- Corrections and additions to the documentation set.

Dylan is a new object-oriented dynamic programming language with support for advanced features such as multiple inheritance, multimethods, and lexical modules. Harlequin Dylan is a powerful integrated development environment for the Windows operating system, including an optimizing compiler, code editor, object browser, debugger, and performance tuning tools.

1 Editions of Harlequin Dylan

There are two editions of Harlequin Dylan: the Personal Edition and the Professional Edition. In general these notes apply to both editions. However, items that apply only to one are flagged as **[Professional Edition]** or **[Personal Edition].** Where there might be ambiguity after two such paragraphs, a return to general material is signalled by **[All Editions]**.

Harlequin Dylan Personal Edition

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

To help foster the widest possible use of the Dylan programming language, Harlequin is making the Personal Edition available for free. But make no mistake about it, although the Personal Edition is free, it not a demo product. It has not been artificially crippled in any way. It is a full Windows development environment.

It is our hope that by using the Personal Edition, many more programmers will come to appreciate the power and the beauty of Dylan.

Harlequin Dylan Personal Edition is available for download from the Harlequin Web site at

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

Documentation is available as a separate download from the same location.

Harlequin Dylan Professional Edition

The Professional Edition is built on the same powerful 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.
- Support for linking to libraries written in C.
- ODBC connectivity libraries.
- 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

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

It can also be purchased by telephone by calling one of the following numbers:

1 (617) 374-2521 (US)
1 (888) 884-8871 (US Toll-Free)
44 (0) 1223 873883 (UK)

After installing the software and familiarizing yourself with the issues covered in the remainder of these notes, your next step is getting started programming in Dylan. We hope that will be a very enjoyable experience for you.

Welcome again to the world of Dylan programming.

2 Installing Dylan

This section outlines the Harlequin Dylan installation procedures and system requirements.

[Professional Edition] The Professional Edition CDROM has an autostart feature. After you insert the CDROM in the drive, just follow the instructions on your screen.

[Personal Edition] The Personal edition is available for downloading from

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

Installation instructions are available on the website.

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

2.1 System Requirements

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

We recommend that you have:

- A minimum of 32 MB of RAM, 64 MB or more preferred.
- Free disk space, as indicated in Table 1:

	Personal Edition	Professional Edition
Target Directory (permanent space)	95 MB	150 - 175 MB
Temporary Space	30 MB	55 MB
Installer Download	30 MB	N/A

Table 1

- A minimum of 128 MB of virtual memory, 160 MB or more preferred.

Note: the temporary directory is chosen according to the value of the **`TMP`** environment variable. If it is not set, then the location for temporary files depends on which operating system you have:

On NT machines, the Windows directory is used.

On Windows95 machines, the current directory is used.

To avoid problems in 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.

- Open a command prompt (**Start > Programs > Command Prompt**).
- Type **`set TMP=`*****path*** where *path* is the path to the temporary directory you want to use (for example, **`d:\tmp`**).

- **[Personal Edition]** Type **`cd`** ***directory*** where *directory* is the directory into which you downloaded the installer for Harlequin Dylan Personal Edition. Then type **`start personal`**.

  [**Professional Edition]** Type **`start`** ***drive*****`:\browser`** where *drive* is the drive letter of your CDROM drive.
- Proceed through the installation process.

2.2 Documentation

[Professional Edition] If you have the Professional Edition of Harlequin Dylan, the installer offers you the choice of installing the documentation as compiled HTML files, raw HTML files, or PDF files. The compiled html can be browsed from within the Dylan Environment using Internet Explorer 3.02 or later. See Section 2.4 on page 6. If you do not want to run Internet Explorer, you may choose the raw HTML files and use your favorite browser, or print out the PDF files. **Note:** the raw HTML files are not browseable from the Dylan environment and do not support global search.

[Personal Edition] If you have the Personal Edition of Harlequin Dylan, the documentation is available as a separate download from:

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

2.3 Reinstalling Harlequin Dylan

To reinstall Harlequin Dylan, you must first remove any existing copies of Harlequin Dylan. Choose **Start > Settings > Control Panels**. Double click on **Add/Remove Programs,** select **Harlequin Dylan**, and click **Add/Remove**. This process does not remove any project files that have been modified since the initial installation. If files or folders are not removed, you can click on the **Details** button to see a list of those files and remove them by hand.

You need to reinstall if you receive a later release of Harlequin Dylan.

[Profession Edition] If you cannot run Harlequin Dylan because of a problem with your serial number and license key, the simplest way to fix it is to reinstall Harlequin Dylan.

2.4 Installing Internet Explorer

To use Harlequin Dylan's online documentation with full browsing/searching capabilities you need Microsoft Internet Explorer 3.02 or later. You can download Internet Explorer from:

`http://www.microsoft.com/ie/download/`

On NT only, Microsoft Internet Explorer 3.02 may need to be installed with administrator privileges.

If you experience problems with installing Microsoft Internet Explorer, please make sure 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, if you are using NT, try reinstalling Microsoft Internet Explorer 3.02 with administrator privileges.

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.

3 Contacting Harlequin Support

[Peraonal Edition] Users of Harlequin Dylan Personal are invited to send questions to the info-dylan mailing list for assistance. You also can access the product knowledgebase at our Support website:

`http://services.harlequin.com/devtools/dylan`

[Professional Edition] Registered users of Harlequin Dylan Professional can contact Harlequin Support via email at:

`dylan-support@harlequin.com`.

Please be certain to include the CD serial number in the subject line of your email message.

If you are not registered, please visit our registration page off our Support website:

`http://services.harlequin.com/devtools/dylan`

You should have your serial number ready. The serial number is located on a sticker on the back of the CD envelope.

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 the 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.

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 The linker

The GNU linker is the default for both editions of Harlequin Dylan.

[Professional Edition] If you have the Professional edition, you have the choice of using the GNU linker or the Microsoft Visual C++ linker. If your project includes C or C++ sources, you must use the Microsoft Visual C++ linker version 4.2 or 5.0.

To use the Microsoft linker, you have to install it from Microsoft Visual C++. 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
```

When you have the Microsoft linker installed from Visual C++, you can use the Microsoft linker by explicitly adding **`/microsoft`** to your command line, or you can set your default linker to the Microsoft linker by doing two things:

- 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 system control panel.
- Ensure the environment has the Visual C++ linker selected. Do this by Selecting **Options > Environment Options...** in the environment window. At the next dialog, select the Linker tab and then choose Microsoft linker.

4.2 Using resources in your project

If you need to compile or recompile any resources, you should be aware that for resource compilation, Harlequin Dylan depends on pre-installation of the Microsoft Platform SDK build-environment. If you do not have the Platform SDK, you can download it from:

```
http://www.microsoft.com/msdn/sdk/sdktools.htm
```

4.3 The compiler

There are two modes of compilation provided by the Dylan compiler, *interactive development mode* and *production mode*. You choose which mode to use from the **Project > Settings** dialog in the environment.

Interactive development mode allows you to redefine forms interactively and incrementally recompile your application. Compilation is faster, but the executable may be slower to run since compile-time type-checking and optimization have been omitted in order to allow the incremental recompilation. Additionally, some warnings may not appear, since type-checking is not rigorous.

Production mode is a fully type-checked and optimized compile. Compilation may take longer, but the executable runs faster.

As the names imply, the interactive development mode is intended for use when you are developing an application and want to be able to change things easily. Production mode is intended for applications that you consider to be fairly complete and ready for final testing and distribution.

4.4 Known compiler problems

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

4.4.1 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.

4.4.2 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 module/libraries.

4.4.3 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**.

4.4.4 Run-time limitations

- **`make`** and **`as`** do not check that their return value has the type of their first argument.
- Unicode is not implemented.
- When comparisons are done between rational and floating numbers, the *Dylan Reference Manual* specifies that 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.
- The Big-Integers library does not provide arbitrary precision integers. Only 64-bit **`<integer>`**s are supported.

- Multiplication does not always signal overflow correctly when using Big-Integers.
- All forms of division (floor/, truncate/, round/, ceiling/) signal an error if both values are outside the range

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

  for the values of **`$minimum-integer`** (2 ^ 29 - 1) and **`$maximum-integer`** (-2 ^ 29) in the Dylan library. So, the following works:

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

  While the following

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

  fails.
- 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 results in an "unexpected end of input" warning from the compiler, and the last form is skipped.
- Console applications (for example, **`factorial-small`** and **`hello-world`**) may not run when invoked from the DOS command line. If attempting to run them from DOS produces the message "This program cannot be run in DOS mode," please make a note of the circumstances (operating system and hardware) as we are trying to better catagorize this problem. You can always double click on these console applications in the Windows Explorer and run them from within the environment.

4.4.5 Compiler limitations

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

- The **`limited`** method on collection classes produces a collection that is limited to specific types. This works for
  `limited(subclass(<sequence>), of:` ***element-type*** **`) =>`**
  `subclass(<vector>)`

  (where *element-type* is **`<byte>`**, **`<integer>`**, **`<double-byte>`**, **`<machine-word>`**, **`<single-float>`**, **`<double-float>`** or **`<object>`**) and for
  `limited(subclass(<stretchy-sequence>), of:` ***element-type*** **`) =>`**
  `subclass(<vector>)`

  (where *element-type* is **`<object>`** or **`<integer>`**). However,

  `limited(subclass(<stretchy-sequence>), of:` ***element-type*** **`)`**

  incorrectly also returns a **`subclass(<vector>)`**, not an **`<array>.`**
- Currently it is not possible to interrupt a compilation (for example, by pressing **Crtl-c**) in the standalone compiler (**`dylan-compile`**) without exiting the compiler.

4.4.6 Garbage collection limitation

Objects referenced only by a weak table are not collected if the weak table is accessed during a GC cycle. If these objects are not collected then their space is not deallocated, and your application's execution time slows down. See Section 4.6.3 on page 13. This is primarily a problem for the Professional Edition, where the ODBC may be accessing tables while the garbage collector, which is asynchronous, is operating.

4.5 Environment problems

- The editor **Options** dialog has a checkbox item "Keep the cursor within the visible area when scrolling with the scrollbar." The checkbox value is ignored and the editor behaves as though it were on.
- When the compiler back-end records debugging information for closure variables, it prefixes their names with the string **`closed-`** as part of uniqueness encoding.
- The environment can get confused if you run more than one Dylan application under the environment at the same time.

4.6 Interoperability

[Professional Edition] This section applies in its entirety to the Professional Edition. It describes the exceptions in the foreign function interfaces to Harlequin Dylan.

4.6.1 Foreign function interface

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 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 Production mode library for use in client libraries that are then free to exploit incremental compilation.

4.6.2 Nonlocal exits

Nonlocal exits from stack frames between Dylan and foreign code should be avoided whenever possible. A nonlocal 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. Nonlocal function exits across the boundary between Dylan and forgeign code are not fully supported. Nonlocal exits between foreign stack frames that pass through Dylan stack frames are not supported at all, and may cause out-of-language errors. Nonlocal exits between Dylan stack frames via foreign stack frames are partially supported. For more information on this, visit the Product Support page on our website.

4.6.3 SQL-ODBC library

Please note the following:

- There is no support for Large Objects in this 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.
- Due to the GC limitation in that objects referenced only by a weak table are not collected if the weak table is accessed during a GC cycle. ODBC resources are not deallocated and you may discover that your application runs progressivly slower and, eventually, crashes. You may see this problem if your application creates a large number of **`<odbc-sql-statement>`** objects on a given connection. The workaround to this problem is to explicitly invoke the **`finalize`** method on the various instances of **`<odbc-sql-statement>`** when they are no longer needed. Result-sets depend on their **`<odbc-sql-statement>`**s and, hence, do not call **`finalize`** on instances of **`<odbc-sql-statement>`** until you are done with the result-set.

4.6.4 OLE, COM, ActiveX, and DBMS documentation errors

Chapter 4 of the *OLE, COM, ActiveX, and DBMS* manual discusses the OLE-Automation library.

The following functions, exported from the OLE-Automation library's OLE-Automation module, are not documented in the reference section of chapter 4: **`pass-as`**, **`out-ref`**, **`inout-ref`**. and **`arg-spec-value`**. Brief descriptions follow.

pass-as *Function*

Summary

Use this in a client as an argument to functions such as **`call-simple-method`** or **`set-property`** or in a server as a return value from a property or method to specify passing the given *value* using the representation designated by *type*.

Signature

```
pass-as (type, value) => (representation :: <ole-arg-spec>)
```

Description

Use this in a client as an argument to functions such as **`call-simple-method`** or **`set-property`** or in a server as a return value from a property or method to specify passing the given *value* using the representation designated by *type*. The *type* may be specified either as one of the low-level VARIANTARG type codes, such as $VT-I2 or $VT-I4 or as a C-FFI type designator, such as **`<C-short>`** or **`<C-long>`**.

out-ref *Function*

Summary

Use this in a client to construct an object that can be passed as an argument to **`call-simple-method`** and receive the value of a returned output parameter.

Signature

```
out-ref (type) => (ref :: <ole-arg-spec>)
```

Description

Use this in a client to construct an object that can be passed as an argument to **`call-simple-method`** and receive the value of a returned output parameter. The *type* of the value is specified as either one of the low-level VARIANTARG type codes (such as $VT-I4) or as a C-FFI type designator, or as a corresponding Dylan type. Use the accessor **`arg-spec-value`** to get the value after the call.

For example, if a server defines a method that has a by-reference output parameter with C type **long**, a Dylan client could receive the value like this:

```
// first make a place to hold the output parameter
let ref = out-ref(<C-long>);
// then call the server method
call-simple-method(disp-interface, disp-id, ref);
// now pick up the received value
let value = arg-spec-value(ref);
```

(If the server is written in Dylan, it will simply receive an instance of **<C-long*>**, and should use **pointer-value-setter** to store the value.)

inout-ref *Function*

Summary

Similar to **out-ref**, except that it is for input-output parameters.

Signature

inout-ref (value, #key vt, type) => (ref :: <ole-arg-spec>)

This is similar to **out-ref** above, except that it is for input-output parameters. The *value* argument is the value to be passed in (may be changed by **arg-spec-value-setter** if the reference is to be used for more than one call), and the type is specified by either the **vt** option with a VARIANTARG type code or the **type:** option with a C or Dylan type designator. If neither **vt**: or **type**: is given, the type will be inferred from the *value*.

The **<ole-arg-spec>** class has the following useful accessor:

arg-spec-value *Function*

Summary

Accessor to return the value from a reference object created by the **out-ref**, **inout-ref**, or **pass-as** functions.

Signature

arg-spec-value (ref :: ***ole-arg-spec*****) => (value ::** ***object*****)**

Description

Accessor to return the value from a reference object created by the **out-ref**, **inout-ref**, or **pass-as** functions. May also be used on the left side of an assignment to change the value.

- Section 4.8 of the *OLE, COM, ActiveX, and DBMS* manual, *Datatypes*, concludes:

  Note that you cannot pass C pointers [between Automation controllers and servers], because the controller and server do not necessarily have access to the same address space. Essentially, what is being passed across the dispinterface is a copy of the value.

  However, for implementing *by-reference* parameter passing, it is permissible to pass certain types of C pointers as argument values, but the server can only use the pointer to load or store a value during the duration of the call. Alternatively, a controller can use the convenience functions **out-ref** and **inout-ref** to implement by-reference parameters without direct handling of C pointers.

  In some cases, a server in another language may require an argument to be passed with a particular representation even though it can't be unambiguously inferred from the actual value. A Dylan client can use the function `**pass-as**' to specify the representation. For example, using the expression **pass-as(<C-long>, 1)** as a method argument forces the value **1** to be passed as a **long** value even though it could fit in a **short**.

- There is an error in the second paragraph of section 4.9, *Memory management issues*, which states:

  If a property value or dispatch method result is a string, it will be represented as an instance of class **<ole-array>** ("BASIC string"), which is a subclass of **<string>** (but not a **<byte-string>**).

  Here, **<ole-array>** should be **<BSTR>**.

- Further in section 4.6.5, *Dispatch method argument and return value types*, note that you can use **`pass-as`** if necessary to constrain the representation of result values.
- In section 4.12, *The OLE-AUTOMATION module*, the reference entry for **`<function-description>`** describes the permitted element types in the sequence that can be passed to **`make`** on **`<function-description>`** with the argument-types: init-keyword. In addition to the C types documented, the sequence can contain instances of **`<ole-arg-spec>`**, as returned by **`out-ref`** or **`inout-ref`**, to designate a *by-reference* parameter.
- The reference entry for **`call-simple-method`**, also in section 4.12, should state that each *args* value can be either the actual value to be passed in the dispatch method call, or an instance of **`<ole-arg-spec>`** from the functions **`pass-as`**, **`out-ref`**, or **`inout-ref`**.

4.7 Harlequin Dylan documentation error

In the Harlequin-Extensions library, the reference page for the **`timing`** macro says that the second result value is in milliseconds (thousandths of a second) when in fact it is in microseconds (millionths of a second)

Index