Harlequin Dylan

Network Library Documentation

Version 1.1

Copyright and Trademarks

Harlequin Dylan: Network Library Documentation

Version 1.1

September 1998

Copyright © 1998 Harlequin Group plc.

Companies, names and data used in examples herein are fictitious unless otherwise noted.

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Harlequin Group plc.

The information in this publication is provided for information only and is subject to change without notice. Harlequin Group plc and its affiliates assume no responsibility or liability for any loss or damage that may arise from the use of any information in this publication. The software described in this book is furnished under license and may only be used or copied in accordance with the terms of that license.

Harlequin is a trademark of Harlequin Group plc.

Dylan is a trademark of Apple Computer, Inc.

Other brand or product names are the registered trademarks or trademarks of their respective holders.

US Government Use

The Harlequin Dylan Software is a computer software program developed at private expense and is subject to the following Restricted Rights Legend: "Use, duplication, or disclosure by the United States Government is subject to restrictions as set forth in (i) FAR 52.227-14 Alt III or (ii) FAR 52.227-19, as applicable. Use by agencies of the Department of Defense (DOD) is subject to Harlequin's customary commercial license as contained in the accompanying license agreement, in accordance with DFAR 227.7202-1(a). For purposes of the FAR, the Software shall be deemed to be `unpublished' and licensed with disclosure prohibitions, rights reserved under the copyright laws of the United States. Harlequin Incorporated, One Cambridge Center, Cambridge, Massachusetts 02142."

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

1 Overview 1

2 Internet addresses 1

Basic Internet address protocol 1
The <IPV6-ADDRESS> class 4
The <NUMERIC-ADDRESS> class 4

3 Sockets 6

The <ABSTRACT-SOCKET> class 6
The <SERVER-SOCKET> class 8
The <TCP-SERVER-SOCKET> class 9
The <SOCKET> class 10
The <BUFFERED-SOCKET> class 11
The <TCP-SOCKET> class 12

4 Socket conditions 13

<socket-condition> 13
<socket-error> 14
<recoverable-socket-condition> 15
<socket-accessor-error> 18

Network Library Reference Documentation

1 Overview

This document covers the Network library. The Network library provides Internet address protocols and TCP/IP server and client sockets. It exports a single module, Network.

The Network library is only available in the Professional and Enterprise editions of Harlequin Dylan.

2 Internet addresses

This section covers Internet address protocols.

2.1 Basic Internet address protocol

This section covers the class **`<internet-address>`** and related generic functions and constants.

<internet-address> *Open abstract primary instantiable class*

Superclasses: **`<object>`**

The class of objects representing Internet addresses used as endpoints for peer-to-peer socket connections.

To construct an **`<internet-address>`** object you must supply either the **`name:`** or **`address:`** keyword. For example:

```
make (<internet-address>, name: "www.whatever.com")
```

or

```
make (<internet-address>,  address: "9.74.122.0")
```

`make` on **`<internet-address>`** returns an instance of **`<ipv4-address>`**.

Keywords:

`name:`	An instance of **`<string>`** representing a symbolic internet address.
`address:`	An instance of **`<string>`** representing a presentation (dotted) form Internet address or an instance of **`<numeric-address>`** (see below).

host-name *Open generic function*

`host-name` ***internet-address*** **`=>`** ***name***

Returns an instance of **`<string>`** containing a symbolic host name. The *internet-address* argument must be an instance of **`<internet-address>`**.

Usually the name returned is the canonical host name. Note, however, that the implementation is conservative about making DNS calls. Suppose that the **`<internet-address>`** instance was created with the **`name:`** keyword and no other information. If the application has not made any other requests that would require a DNS call, such as to **`host-address`** or **`aliases`** (see below), the name that this function returns will be the one specified with the **`name:`** keyword, regardless of whether that is the canonical name or not.

host-address *Open generic function*

`host-address` ***internet-address*** **`=>`** ***address***

Returns an instance of **`<string>`** containing the presentation form of the host address. In the case of multi-homed hosts this will usually be the same as:

```
multi-homed-internet-address.all-addresses.first.host-address
```

In the case of an Internet address created using the **`address:`** keyword it will be either the keyword value or **`all-addresses.first.host-address`**.

numeric-host-address *Open generic function*

`numeric-host-address` ***internet-address*** **`=>`** ***numeric-address***

Returns the host address as a **`<numeric-address>`**.

all-addresses *Open generic function*

`all-addresses` ***internet-address*** **`=>`** ***sequence***

Returns an instance of **`<sequence>`** whose elements are **`<internet-address>`** objects containing all known addresses for the host.

aliases *Open generic function*

`aliases` ***internet-address*** **`=>`** ***sequence***

Returns an instance of **`<sequence>`** whose elements are instances of **`<string>`** representing alternative names for the host.

$loopback-address *Constant*

An instance of **`<internet-address>`** representing the loopback address: **`"127.0.0.1"`**.

$local-host *Constant*

An instance of **`<internet-address>`** representing the host on which the application using sockets is correctly running.

Note that this value is not necessarily the same as would be created by the expression

```
make (<internet-address>, name: "localhost")
```

The address assigned to the symbolic name **`localhost`** is dependent on the configuration of DNS. In some cases this may be configured to be the loopback address rather than a real address for the local host.

2.2 The <IPV6-ADDRESS> class

This name is reserved for future development.

2.3 The <NUMERIC-ADDRESS> class

This section describes numeric Internet representation and associated protocols.

<numeric-address> *Sealed abstract primary class*

Superclasses: **`<object>`**

The class of objects representing the numeric form of an Internet addresses.

Currently only ipv4 (32-bit) addresses are supported. Ipv6 addresses will be added when they are supported by Winsock2. In general **`<numeric-address>`** objects are accessed using the functions **`host-order`** or **`network-order`**, depending on the context in which they are employed.

network-order *Sealed generic function*

`network-order` ***address*** **`=>`** ***network-order-address***

Returns the value of the numeric address in network order. The argument is a general instance of **`<numeric-address>`**. The class of the object returned depends upon the particular subclass of the argument; the **`network-order`** method for **`<ipv4-numeric-address>`** returns an instance of **`<machine-word>`**.

Network order is big-endian byte order.

host-order *Sealed generic function*

`host-order` ***address*** **`=>`** ***host-order-address***

Like **`network-order`** but returns the value in host order.

Host order is either big-endian byte order on a big-endian host machine and little-endian on a little-endian host machine.

2.3.1 IPV4 addresses

<ipv4-numeric-address> *Open abstract primary instantiable class*

Superclasses: **`<numeric-address>`**

The single slot of this class contains a 32-bit value representing a ipv4 address. This slot is accessed by the generic functions **`network-order`** and **`host-order`** described above. **`<ipv4-numeric-address>`** has two concrete subclasses **`<ipv4-network-order-address>`** and **`<ipv4-host-order-address>`**. Make **`<ipv4-numeric-address>`** returns one or the other of these depending upon the value of the **`order:`** keyword.

Keywords:

`value:`	An instance of **`<machine-word>`**. Required.
`order:`	One of **`#"network-order"`** or **`#"host-order"`**. Required.

host-order *G.f. method*

`host-order` ***ip4-numeric-address*** **`=>`** ***machine-word***

Returns the numeric address in host order as an instance of **`<machine-word>`**. The argument is an instance of **`<ip4-numeric-address>`**.

network-order *G.f. method*

`network-order <ipv4-numeric-address> => <machine-word>`

Returns the numeric address in network order as an instance of **`<machine-word>`**. The argument is an instance of **`<ip4-numeric-address>`**.

as *G.f. method*

`as` ***string ipv4-numeric-address*** **`=>`** ***string***

Returns the presentation (dotted string) form of an instance of **`<ip4-numeric-address>`**.

<ipv4-network-order-address> *Sealed concrete class*

Superclasses: **`<ipv4-numeric-address>`**

Concrete subclass for network-order numeric addresses.

```
make(<ipv4-network-order-address>)
```

is equivalent to

```
make(<ipv4-numeric-address>, order: network-order)
```

<ipv4-host-order-address> *Sealed concrete class*

Superclasses: **`<ipv4-numeric-address>`**

Concrete subclass for host order numeric addresses.

3 Sockets

This section describes socket classes and protocols.

3.1 The <ABSTRACT-SOCKET> class

<abstract-socket> *Open abstract uninstantiable free class*

Superclasses: **`<object>`**

The common superclass of all socket objects including **`<socket>`** (IP client socket), **`<server-socket>`** and **`<socket-accessor>`**.

Keywords:

`socket-descriptor:`

A Windows handle or UNIX fd (file descriptor) for the socket. In general users of the sockets API should not need to use this keyword. Only implementors of new socket classes should be interested.

Each subclass of **`<abstract-socket>`** must provide methods for **`close`** and for the following generic functions:

local-port *Open generic function*

`local-port` ***socket*** **`=>`** ***port-number***

Returns the local port number for an instance of **`<socket>`**, **`<datagram-socket>`** or **`<server-socket>`**. The return value is an instance of **`<integer>`**.

socket-descriptor *Open generic function*

`socket-descriptor` ***socket*** **`=> descriptor`**

Returns the descriptor (handle or fd) for the socket. The argument is an instance of **`<abstract-socket>`** and the return value an instance of **`<accessor-socket-descriptor>`**.

local-host *Open generic function*

`local-host` ***socket*** **`=>`** ***host-address***

Returns the address of the local host. The argument is an instance of **`<abstract-socket>`** and the return value an instance of **`<internet-address>`**.

3.2 The <SERVER-SOCKET> class

<server-socket> *Open abstract primary instantiable class*

Superclasses: **`<abstract-socket>`**

Server-sockets listen on a specified port for connection requests which come in over the network. Either the **`port:`** or **`service:`** keyword must be supplied.

Keywords:

`service:`	An instance of **`<string>`** containing an abstract name for a service with a “well-known” port, such as **`"ftp"`** or **`"daytime"`**. Valid names depend on the configuration of the DNS. Required unless **`port:`** is supplied.
`port:`	An instance of **`<integer>`** identifying the port on which the **`<server-socket>`** should listen for connection requests. Required unless **`service:`** is supplied.
`protocol:`	An instance of **`<string>`** naming the protocol. Currently **`"tcp"`** is the only supported protocol. You can create instances of protocol-specific subclasses as an alternative to using the **`protocol:`** keyword. For example, **`make(<server-socket>, protocol: "tcp", …)`** is equivalent to **`make(<TCP-server-socket>, …)`**. **`make`** on **`(<server-socket>)`** returns an instance of **`<tcp-server-socket>`** by default.

accept *Open generic function*

`accept` ***server-socket*** **`#key =>`** ***result***

Blocks until a connect request is received. Returns a connected instance of **`<socket>`**. The particular subclass of **`<socket>`** returned depends on the actual class of the argument, which must be a general instance of

`<server-socket>`. Calling **`accept`** on **`<tcp-server-socket>`** returns a connected **`<tcp-socket>`**.

with-server-socket *Macro*

```
with-server-socket (server-var [:: server-class], keywords)
  body
end;
```

Creates an instance of **`<server-socket>`**, using the (optional) *server-class* argument and keyword arguments to make the **`<server-socket>`**, and binds it to the local variable named by *server-var*. The *body* is evaluated in the context of the binding and the **`<server-socket>`** is closed after the body is executed.

start-server *Macro*

```
start-server ([server-var = ]socket-server-instance,
              socket-var [, keywords])
  body
end;
```

Enters an infinite **`while(#t) accept`** loop on the server socket. Each time accept succeeds the **`<socket>`** returned from accept is bound to *socket-var* and the *body* is evaluated in the context of the binding. When *body* exits, **`accept`** is called again producing a new binding for *socket-var*. The optional keywords are passed to the call to **`accept`**.

3.3 The <TCP-SERVER-SOCKET> class

<tcp-server-socket> *Concrete primary sealed class*

Superclass: **`<server-socket>`**

The class of TCP server sockets. A server socket is an object which listens for requests for connections from the network. When accept is called on the server socket and a request for connection is detected, accept returns a connected **`<socket>`**.

Keywords:

`element-type:`	Establishes a new default for the **`element-type`** of **`<TCP-socket>`** instances returned by calling **`accept`** with this server socket as the argument to **`accept`**. This default **`element-type`** may be overridden for any particular call to **`accept`** by using the **`element-type:`** keyword to **`accept`**. If no **`element-type:`** is specified when the server socket is created, **`<byte-character>`** is used as the default **`element-type`**.

accept *G.f. method*

`accept` ***server-socket*** **`:: <tcp-server-socket>, #key`** ***element-type*** **`=>`** ***connected-socket***

This method on **`accept`** returns a connected instance of **`<tcp-socket>`**. The **`element-type:`** keyword controls the element type of the **`<tcp-socket>`** (stream) returned from **`accept`**. If the keyword is not supplied, the default value used is **`#f`**.

3.4 The <SOCKET> class

socket *Open abstract free instantiable class*

Superclasses: **`<abstract-socket>`**, **`<external-stream>`**

The class of general client sockets. All client sockets are streams.

Keywords:

`direction:`	Specifies the direction of the stream. It must be one of **`#"input"`**, **`#"output"`**, and **`"#input-output"`**. This keyword is an inherited streams class keyword. See the Streams library documentation in the *System and I/O* library reference for a full description.

`element-type:` An instance of **`<class>`**. Useful values are **`<byte-character>`** and **`<byte>`**. This keyword is an inherited streams class keyword. See the Streams library documentation in the *System and I/O* library reference for a full description.

3.5 The <BUFFERED-SOCKET> class

<buffered-socket> *Open abstract free class*

Superclasses: **`<socket>`**, **`<double-buffered-stream>`**

Socket streams whose elements are bytes or characters. These inherit buffering protocols and the implications of **`read`**, **`write`**, **`read-element`**, **`write-element`**, **`force-output`** and suchlike methods from **`<double-buffered-stream>`**.

Keywords:

`force-output-before-read?:`

An instance of **`<boolean>`**. Defaults value: **`#t`**. The methods which implement the stream reading protocols (**`read`**, **`read-line`**, **`read-element`** and so on) for instances of **`<socket>`** call **`force-output`** by default before blocking. This is to ensure that any pending output has been sent to the peer before the socket blocks waiting to read data sent by the peer. This corresponds to the expected, usual behavior of single-threaded client sockets and avoids deadlock in usual cases. Multi-threaded applications, particularly applications where one thread is reading and another thread is writing to the same socket, may wish to inhibit the default **`force-output`**. If the socket is created with **`force-output-before-read?:`** as **`#f`**, **`force-output`** will not be called before the read functions block.

3.6 The <TCP-SOCKET> class

The class of TCP client sockets.

<tcp-socket> *Abstract primary sealed class*

Superclasses: **`<buffered-socket>`**

The class of TCP client sockets.

Keywords:

Of the keywords below, **`host:`** and one of either **`service:`** or **`port:`** are required.

`host:`	An instance of **`<internet-address>`** or **`<string>`**. The remote host to connect to. The **`<string>`** may be either a host name or a presentation-form Internet address. Required.
`service:`	An instance of **`<string>`**. A **`<string>`** containing an abstract name for a service with a "well-known" port, such as **`"ftp"`** or **`"daytime"`**. Valid names depend on the configuration of the DNS. Required unless **`port:`** is supplied.
`protocol:`	An instance of **`<string>`** naming the protocol. Currently **`"tcp"`** is the only supported protocol. You can create instances of protocol-specific subclasses as an alternative to using the **`protocol:`** keyword. For example **`make(<socket>, protocol: "tcp", …)`** is equivalent to **`make(<TCP-socket>, …)`**. **`make`** on **`<socket>`** returns an instance of **`<tcp-server-socket>`** by default.
`port:`	An instance of **`<integer>`** representing the remote port to connect to. Required unless **`service:`** is supplied.

`element-type:`	An instance of **`<class>`**. Useful values for **`<tcp-streams>`** are **`<byte-character>`** and **`<byte>`**. This keyword is an inherited streams class keyword. See the Streams library documentation in the *System and I/O* library reference for a full description.

remote-port *Open generic function*

`remote-port` ***socket*** **`=>`** ***port-number***

Returns the remote port number for a **`<socket>`**. The value returned is an instance of **`<integer>`**.

remote-host *Open generic function*

`remote-host` ***socket*** **`=>`** ***remote-host-address***

Returns the remote host for a **`<socket>`**. The value returned is an instance of **`<internet-address>`**.

4 Socket conditions

This section lists the socket condition classes in the Network library.

4.1 <socket-condition>

All socket conditions are packaged into general instances of **`<socket-condition>`**. Some conditions are considered recoverable and others not.

<socket-condition> *Condition*

Superclasses: **`<simple-condition>`**

The class of socket conditions. It inherits the **`format-string:`** and **`format-arguments:`** keywords from **`<simple-condition>`**.

Slots:

`socket-condition-details`

Most socket conditions originate in error return codes from Harlequin Dylan’s Winsock2 library, an FFI interface to the native socket library Winsock2.

The **`socket-condition-details`** slot provides information about the low-level failure which was the source for the condition. In most cases this slot will hold an instance of **`<socket-accessor-condition>`**, below.

When creating general instances of **`<socket-condition>`**, you can use the **`details:`** keyword to set the value for this slot.

4.2 <socket-error>

The class **`<socket-error>`** is the superclass of all unrecoverable socket conditions.

<socket-error> *Condition*

Superclasses: **`<socket-condition>`**

The class of socket conditions from which no recovery is possible.

4.2.1 <internal-socket-error>

The class **`<internal-socket-error>`** is the class of unexpected socket errors.

<internal-socket-error> *Condition*

Superclasses: **`<socket-error>`**

The class of unexpected errors from Harlequin Dylan’s Winsock2 library, an FFI interface to the native socket library Winsock2.

Inspect the contents of the **`socket-condition-details`** slot for more information.

4.3 <recoverable-socket-condition>

The **`<recoverable-socket-condition>`** class is the general class of socket conditions for which an application may be able to take some remedial action.

<recoverable-socket-condition> *Condition*

Superclasses: **`<socket-condition>`**

The general class of socket conditions for which an application may be able to take some remedial action.

For instance, a web browser receiving such conditions as **`<connection-refused>`** or **`<host-not-found>`** (see below) would normally expect to report those conditions to the user and continue with some other connection request from the user, while a server receiving a **`<connection-closed>`** condition from a connected **`<socket>`** would probably close the **`<socket>`** and continue to handle other requests for connections.

4.3.1 <network-not-responding>

<network-not-responding> *Condition*

Superclasses: **`<recoverable-socket-condition>`**

The network — probably a local network — is down. Try again later.

4.3.2 <invalid-address>

<invalid-address> *Condition*

Superclasses: **`<recoverable-socket-condition>`**

A badly formed address string has been passed to a function trying to make an **`<internet-address>`**.

4.3.3 <host-not-found>

<host-not-found> *Condition*

Superclasses: **`<recoverable-socket-condition>`**

The Domain Name Server (DNS) cannot resolve the named host or internet address. Try again with a different (correct) name or address.

4.3.4 <server-not-responding>

<server-not-responding> *Condition*

Superclasses: **`<recoverable-socket-condition>`**

The Domain Name Server (DNS) did not respond or returned an ambiguous result. Try again.

4.3.5 <host-unreachable>

<host-unreachable> *Condition*

Superclasses: **`<recoverable-socket-condition>`**

The remote host cannot be reached from this host at this time.

4.3.6 <socket-closed>

<socket-closed> *Condition*

Superclasses: **`<recoverable-socket-condition>`**

The socket or server socket has been closed.

Most operations on closed instances of **`<TCP-socket>`** return instances of **`<stream-closed-error>`** (from the Streams library) rather than instances of **`<socket-closed>`**.

4.3.7 <connection-failed>

<connection-failed> *Condition*

Superclasses: **`<recoverable-socket-condition>`**

The attempt to connect to the remote host was not successful. Connection failed for one of the following reasons: because the connect request timed out or because it was refused, or because the remote host could not be reached.

4.3.8 <connection-closed>

<connection-closed> *Condition*

Superclasses: **`<recoverable-socket-condition>`**

The connection to the remote host has been broken. The socket should be closed. To try again, open a new socket.

4.3.9 <address-in-use>

<address-in-use> *Condition*

Superclasses: **`<recoverable-socket-condition>`**

A process on the machine is already bound to the same fully qualified address. This condition probably occurred because you were trying to use a port with an active server already installed, or a process crashed without closing a socket.

4.3.10 <blocking-call-interrupted>

<blocking-call-interrupted> *Condition*

Superclasses: **`<recoverable-socket-condition>`**

A blocking socket call, like **`read`**, **`write`** or **`accept`**, was interrupted.

4.3.11 <out-of-resources>

<out-of-resources> *Condition*

Superclasses: **`<recoverable-socket-condition>`**

The implementation-dependent limit on the number of open sockets has been reached. You must close some sockets before you can open any more. The limits for Windows NT (non-server machines) and Windows 95 are particularly small.

4.4 <socket-accessor-error>

<socket-accessor-error> *Condition*

Superclasses: **`<socket-error>`**

An implementation-specific error from the C-FFI interface to the native socket library. Usually instances of this class these appear in the **`socket-condition-details`** slot of another **`<socket-condition>`**.

4.4.1 <win32-socket-error>

<win32-socket-error> *Condition*

Superclasses: **`<socket-accessor-error>`**

A Win32-specific error from the Winsock2 library, a C-FFI interface to the native socket library Winsock2. A function in the FFI library has returned an error return code.

Slots:

`WSA-numeric-error-code`

Contains the numeric error code that was returned. An instance of **`<integer>`**.

`WSA-symbolic-error-code`	Contains an instance of **`<string>`** giving the symbolic (human-readable) form of the error code. For example, the string might be **`"wsanotsock"`**.
`explanation`	An explanation if any of the error. An instance of **`<string>`**.
`calling-function`	The name of Winsock2 FFI interface function which returned the error code. An instance of **`<string>`**.