At Harlequin created on August 31, 1995 and last modified at Harlequin on September 7, 1995.

DylanWorks

Standard I/O Libraries

Scott McKay

1. Introduction

This document is intended to give an overview of the various standard I/O libraries, including:

- Streams, which includes support for streams over sequences and files.
- "Standard I/O", which includes support for Lisp-style ***standard-input*** and ***standard-output***.
- **format** and **print-object**.
- Locators.
- File system management.

2. Streams

The purposes of the Dylan Streams Library are:

- To provide a generic, easy-to-use interface for streaming over sequences and files. The same interface for consuming or producing is available irrespective of the type of stream, or the types of the elements being streamed over.
- To provide an efficient system, especially for the common case of file I/O.
- To provide access to an underlying buffer management protocol.
- Random access to file streams.
- To provide an extensible framework. Other useful functionality that sits behind a stream interface should be easy to integrate to this stream library.
- Support for "wrapper" streams.

The Streams library is not intended to address the following:

- Complex streams, such as "broadcast" streams. This functionality can be easily provided by other, layered libraries.
- A standard object-printing package such as Smalltalk's `printOn:` or Lisp's `print-object`, or a formatted printing facility such as Lisp's `format`. This is addressed by the Format library.
- General object dumping and loading.

- A comprehensive range of I/O facilities for using memory-mapped files, network connections, etc., although it is expected that such facilities can be easily added to the streams library by virtue of its extensible framework.
- Audio, multi-media, and soon.
- An interface to operating system functionality such as pathnames, system calls, etc. These are addressed by the Locators and File System libraries.

Note that what we are calling the “Streams Library” may could be implemented as several libraries, for example, one that implements support for streams over internal sequences, another that supports streams over files, another that supports wrapper streams, and so on. This is addressed in the Streams Library Proposal.

3. Standard I/O

Fill this in

4. Format and Object printing

Fill this in

5. Locators

The purpose of the Dylan Locators library is to provide Dylan programs with a portable, flexible, and uniform facility for manipulating the “addresses” of files, databases, and other network resources on a variety of platforms. The object that provides this service is a *locator*.

For programs which must be ported across operating systems and for programs which need to be installed on many machines, code that explicitly names external resources (such as files and peripheral devices) can be a barrier to portability.

This library addresses these issues in two ways:

- It provides a portable and reliable means of parsing file names and World-Wide Web Uniform Resource Locators (URLs) and manipulating them as though they were structured objects.
- It provides a portable, operating system-independent, “abstract” file system which can be configured to map onto one or more physical file systems.

This library does not address the issue of locating resources which are linked into a program or which are accessible using operating system-specific APIs. For example, Apple’s MacOS provides an elaborate scheme for naming and referencing fonts, bitmaps, menus, and so forth. This is beyond the scope of this library. This library also does not address issues of accessing file contents or operating on file systems. This functionality is provided by the Dylan Streams Library and the Dylan File System Library.

The pathname library provides the following functionality:

- Parsing strings using either the native filename syntax or URL syntax into locator objects.
- Coercing locator objects back into strings.
- Locator merging, that is, filling in the unspecified components of one locator from defaults supplied by another locator, and resolving "relative" locators into "absolute" locators. Locators having different characteristics (such as different syntaxes, coming from different file systems, and so on) can be meaningfully merged, for example, merging the Unix pathname `/usr/local/bin/` against the Macintosh pathname `disk:system:network:TCP` will produce the Unix pathname `/usr/local/bin/TCP`.
- Abstract locators, which are a kind of locator that are mapped to a "physical" locators by a set of site-dependent translation rules, and canonical file types (for example, `.TXT` and `.TEXT` might have the same canonical type, **#"text"**).
- Support for versioned file systems, including support for naming "branches" and versions of source files in source control systems such as RCS.
- Enumerating a set of files that match a given input pattern, for purposes of wildcard matching and "completion".
- Generation of temporary file names.
- Specific requirements for behavior on each OS (e.g., Mac, Microsoft, Unix).
- An extensible protocol for supporting new types of host pathnames.

6. File System Library

The Dylan File System Library provides the following functionality:

- Finding the current "home" directory.
- Renaming and deleting files.
- Reading and setting file properties (such as the file author, creation date, access control lists, and so on).