21. Streams

21.1 Stream Concepts

21.1.1 Introduction to Streams

A *stream* is an *object* that can be used with an input or output function to identify an appropriate source or sink of *characters* or *bytes* for that operation. A *character stream* is a source or sink of *characters*. A *binary stream* is a source or sink of *bytes*.

Some operations may be performed on any kind of *stream*; the next figure provides a list of *standardized* operations that are potentially useful with any kind of *stream*.

```
close                 stream-element-type
input-stream-p        streamp
interactive-stream-p  with-open-stream
output-stream-p
```

Figure 21-1. Some General-Purpose Stream Operations

Other operations are only meaningful on certain *stream types*. For example, **read-char** is only defined for *character streams* and **read-byte** is only defined for *binary streams*.

21.1.1.1 Abstract Classifications of Streams

21.1.1.1.1 Input, Output, and Bidirectional Streams

A *stream*, whether a *character stream* or a *binary stream*, can be an *input stream* (source of data), an *output stream* (sink for data), both, or (e.g., when "`:direction :probe`" is given to **open**) neither.

The next figure shows *operators* relating to *input streams*.

```
clear-input  read-byte           read-from-string
listen       read-char           read-line
peek-char    read-char-no-hang   read-preserving-whitespace
read         read-delimited-list unread-char
```

Figure 21-2. Operators relating to Input Streams.

The next figure shows *operators* relating to *output streams*.

```
clear-output   prin1           write
finish-output  prin1-to-string write-byte
force-output   princ           write-char
format         princ-to-string write-line
fresh-line     print           write-string
pprint         terpri          write-to-string
```

Figure 21-3. Operators relating to Output Streams.

A *stream* that is both an *input stream* and an *output stream* is called a *bidirectional stream*. See the *functions* **input-stream-p** and **output-stream-p**.

Any of the *operators* listed in Figure 21-2 or Figure 21-3 an be used with *bidirectional streams*. In addition, the next figure hows a list of *operators* that relate specificaly to *bidirectional streams*.

```
y-or-n-p  yes-or-no-p
```

Figure 21-4. Operators relating to Bidirectional Streams.

21.1.1.1.2 Open and Closed Streams

Streams are either *open* or *closed*.

Except as explicitly specified otherwise, operations that create and return *streams* return *open streams*.

The action of *closing* a *stream* marks the end of its use as a source or sink of data, permitting the *implementation* to reclaim its internal data structures, and to free any external resources which might have been locked by the *stream* when it was opened.

Except as explicitly specified otherwise, the consequences are undefined when a *closed stream* is used where a *stream* is called for.

Coercion of *streams* to *pathnames* is permissible for *closed streams*; in some situations, such as for a *truename* computation, the result might be different for an *open stream* and for that same *stream* once it has been *closed*.

21.1.1.1.3 Interactive Streams

An *interactive stream* is one on which it makes sense to perform interactive querying.

The precise meaning of an *interactive stream* is *implementation-defined*, and may depend on the underlying operating system. Some examples of the things that an *implementation* might choose to use as identifying characteristics of an *interactive stream* include:

* The *stream* is connected to a person (or equivalent) in such a way that the program can prompt for information and expect to receive different input depending on the prompt.
* The program is expected to prompt for input and support "normal input editing".
* **read-char** might wait for the user to type something before returning instead of immediately returning a character or end-of-file.

The general intent of having some *streams* be classified as *interactive streams* is to allow them to be distinguished from streams containing batch (or background or command-file) input. Output to batch streams is typically discarded or saved for later viewing, so interactive queries to such streams might not have the expected effect.

Terminal I/O might or might not be an *interactive stream*.

21.1.1.2 Abstract Classifications of Streams

21.1.1.2.1 File Streams

Some *streams*, called *file streams*, provide access to *files*. An *object* of *class* **file-stream** is used to represent a *file stream*.

The basic operation for opening a *file* is **open**, which typically returns a *file stream* (see its dictionary entry for details). The basic operation for closing a *stream* is **close**. The macro **with-open-file** is useful to express the common idiom of opening a *file* for the duration of a given body of *code*, and assuring that the resulting *stream* is closed upon exit from that body.

21.1.1.3 Other Subclasses of Stream

The *class* **stream** has a number of *subclasses* defined by this specification. The next figure shows some information about these subclasses.

Class	Related Operators
broadcast-stream	make-broadcast-stream
	broadcast-stream-streams
concatenated-stream	make-concatenated-stream
	concatenated-stream-streams
echo-stream	make-echo-stream
	echo-stream-input-stream
	echo-stream-output-stream
string-stream	make-string-input-stream
	with-input-from-string
	make-string-output-stream
	with-output-to-string
	get-output-stream-string
synonym-stream	make-synonym-stream
	synonym-stream-symbol
two-way-stream	make-two-way-stream
	two-way-stream-input-stream
	two-way-stream-output-stream

Figure 21-5. Defined Names related to Specialized Streams

21.1.2 Stream Variables

Variables whose *values* must be *streams* are sometimes called *stream variables*.

Certain *stream variables* are defined by this specification to be the proper source of input or output in various *situations* where no specific *stream* has been specified instead. A complete list of such *standardized stream variables* appears in the next figure. The consequences are undefined if at any time the *value* of any of these *variables* is not an *open stream.*

Glossary Term	Variable Name
debug I/O	*debug-io*
error output	*error-output*
query I/O	*query-io*
standard input	*standard-input*
standard output	*standard-output*
terminal I/O	*terminal-io*
trace output	*trace-output*

Figure 21-6. Standardized Stream Variables

Note that, by convention, *standardized stream variables* have names ending in "`-input*`" if they must be *input streams*, ending in "`-output*`" if they must be *output streams*, or ending in "`-io*`" if they must be *bidirectional streams*.

User programs may *assign* or *bind* any *standardized stream variable* except ***terminal-io***.

21.1.3 Stream Arguments to Standardized Functions

The *operators* in the next figure accept *stream arguments* that might be either *open* or *closed streams*.

```
broadcast-stream-streams     file-author       pathnamep
close                        file-namestring   probe-file
compile-file                 file-write-date   rename-file
compile-file-pathname        host-namestring   streamp
concatenated-stream-streams  load              synonym-stream-symbol
delete-file                  logical-pathname  translate-logical-pathname
directory                    merge-pathnames   translate-pathname
directory-namestring         namestring        truename
dribble                      open              two-way-stream-input-stream
echo-stream-input-stream     open-stream-p     two-way-stream-output-stream
echo-stream-ouput-stream     parse-namestring  wild-pathname-p
ed                           pathname          with-open-file
enough-namestring            pathname-match-p
```

Figure 21-7. Operators that accept either Open or Closed Streams

The *operators* in the next figure accept *stream arguments* that must be *open streams*.

```
clear-input               output-stream-p          read-char-no-hang
clear-output              peek-char                read-delimited-list
file-length               pprint                   read-line
file-position             pprint-fill              read-preserving-whitespace
file-string-length        pprint-indent            stream-element-type
finish-output             pprint-linear            stream-external-format
force-output              pprint-logical-block     terpri
format                    pprint-newline           unread-char
fresh-line                pprint-tab               with-open-stream
get-output-stream-string  pprint-tabular           write
input-stream-p            prin1                    write-byte
interactive-stream-p      princ                    write-char
listen                    print                    write-line
make-broadcast-stream     print-object             write-string
make-concatenated-stream  print-unreadable-object  y-or-n-p
make-echo-stream          read                     yes-or-no-p
make-synonym-stream       read-byte
make-two-way-stream       read-char
```

Figure 21-8. Operators that accept Open Streams only

21.1.4 Restrictions on Composite Streams

The consequences are undefined if any *component* of a *composite stream* is *closed* before the *composite stream* is *closed*.

The consequences are undefined if the *synonym stream symbol* is not *bound* to an *open stream* from the time of the *synonym stream*'s creation until the time it is *closed*.