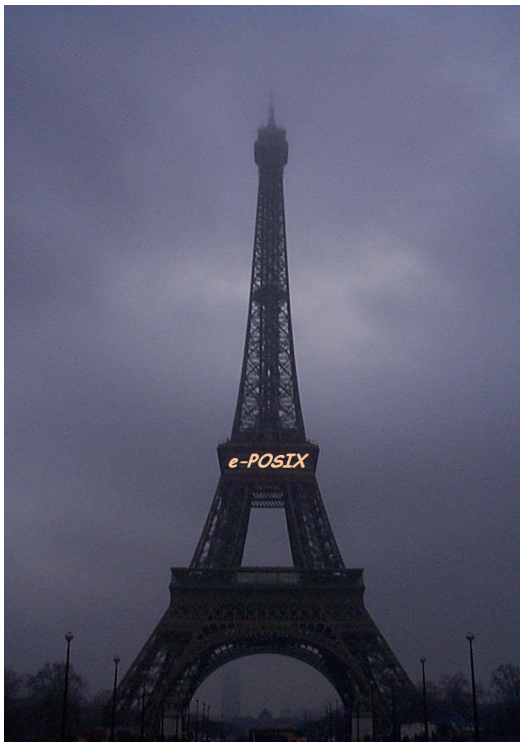

e-POSIX

The definitive and complete Eiffel to Standard C and POSIX 1003.1 binding

written by Berend de Boer

Contents

Introduction

It has been a great pleasure for me when I could announce the first public alpha release of this manual. And then came the betas and the first release. Writing libraries like this is boring stuff. Every Eiffel programmer should have had access to all those Standard C and POSIX routines long ago. Anyway, now you and me have. Whatever a C programmer can do, you can. And even more safe as this library protects you of inadvertently calling routines that are not portable (because they're simply not there :-)).

Writing libraries like this also seems to be a never ending story, as we now are at version 1.5. And my to do list hasn't shrinked, so stay tuned!

I will support this library, so bug reports and wishes are gladly accepted. In the future, I hope to be able to expand this library to add more stuff from the Open Unix Specification, particularly sockets and curses. Perhaps the authors of the existing Eiffel implementations for these APIs are willing to create one single unified library.

Have fun using this library and I like to hear about applications!

Licensing

This software is licensed under the Eiffel Forum Freeware License, version 1. This license can be found in the `forum.txt` file. Basically this license allows you to do anything with it, i.e. use it for commercial or Open Source software without restrictions. But don't sue me if something goes wrong. And give me some credits.

Also explicitly allowed is copying parts of this library to your own, for example copying certain Standard C or POSIX header wrappings. I prefer linking, but you don't have to retype everything if you don't want to link.

Support

e-POSIX is a fully supported program. You can send requests for help directly to me. But to help others profit from the discussion, and perhaps to get feedback when I'm short on time, it is suggested that support messages are sent to **eposix@yahoogroups.com**.

Latest versions and announcements are available from `http://groups.yahoo.com/group/eposix/`.

Commercial support

I'm available to give companies or organisations a one or two day course using POSIX and in particularly this library. Prices are $1000 NZD a day, excluding VAT, travel and hotel expenses. Contact me at **berend@pobox.com**.

Acknowledgements

I like to thank people who, one way or another, have helped me in creating this library. They're listed in order they have been involved with this library or manual:

- **Eugene Melekhov <eugene_melekhov@object-tools.com>**: compiled it with Visual Eiffel. As Visual Eiffel is the most strict compiler, he found a great many oversights that SmallEiffel didn't catch.
- **mico/E team**: I got many ideas for my C interface from the mico/E project. Sometime ago **Andreas Schulz** wrote me that the micoe team wanted to use e-POSIX in mico/E. Andreas also reported problems and suggested improvements, especially in the *EPX_CGI* class. And he continues to send bug reports, thanks!
- **Ida de Boer <ida@gameren.nl>**: it was she who provided you with the POSIX to Eiffel mapping table in **appendix A**.
- **Steve Harris <scharris@worldnet.att.net>**: suggested improvements, found a CAT call problem and we had an interesting discussion about forking.
- **Jörgen Tegnér <teg@post.netlink.se>** reported a problem with an example, and a bug in *POSIX_EXEC_PROCESS*.
- **Marcio Marchini <mqm@magma.ca>** contributed a lot to e-POSIX. He gave very useful advice, submitted code, and supplied patches to compile e-POSIX better on Windows. I think it is fair to say that you thank the Windows support in e-POSIX to Marcio.
- **Eric Bezault**: I've had some insightful discussions with Eric regarding architecture of libraries such as e-POSIX. I think we never agreed :-), but the alternative error handling is due to his comments!
- **Andreas Leitner**: Discussions about using eposix which will lead to even closer integration with Gobo in subsequent releases.

Colophon

The text of this manual was entered with GNU Emacs 20.7.1 on RedHat Linux 7.1. It was typeset with pdfTEX using the ConTEXt macro package, see `http://www.pragma-ade.com`. BON diagrams were created with METAPOST.

In this chapter:

- ***Requirements***
- ***Compiling the C code***

1 Requirements and installation

1.1 Requirements

e-POSIX has three requirements:

1. e-POSIX requires Gobo release 3.0 or higher. You can download Gobo at `http://www.gobosoft.com/`. Gobo must be installed.
2. e-POSIX requires that the environment variable `EPOSIX` is set to the root directory where the e-POSIX are unpacked.
3. On Windows, e-POSIX requires that the environment variable `GOBO_CC` is set to the name of the C compiler you are using. Failure to do so will result in link errors. Perhaps in a future `geant` release this will be set automatically.

1.2 Compiling the C code

Before e-POSIX can be used, a few C files need to be compiled into a library. The steps differ if you are using a Unix derivative, or a Windows based system.

1.2.1 Compiling on Unix

Before the C files can be compiled, e-POSIX must be configured. If you have just one Eiffel compiler on your system, this should be sufficient:

```
./configure --prefix=$EPOSIX
make
```

If you have multiple Eiffel compilers, you can specify the compiler with:

```
./configure --with-compiler=ve --prefix=$EPOSIX
```

The `--prefix` switch is a trick to make sure that you can type:

```
make install
```

after the make was successful. With this step the library is installed into the `\$EPOSIX/lib` directory. This is the location where e-POSIX's `src/library.xace` expects it. Without the `--prefix` switch the library will usually be installed in `/usr/local/lib`.

More information about `configure` options can be displayed with:

```
./configure --help
```

1.2.2 Compiling on Windows

For Windows system, I've supplied a tool —build with e-POSIX— that can build the necessary e-POSIX library for your Eiffel and C compiler.

Type:

```
makelib
```

to get help. Type:

```
makelib -ise -msc
```

to compile the C code with Microsoft's Visual C compiler targeting the ISE Eiffel compiler.

Only the Microsoft supplied library did work, i.e. link, with VisualEiffel:

```
makelib -ve -msc
```

Type:

```
makelib -se -bcc
```

to compile the C code with Borland's C compiler targeting SmallEiffel. It was tested with the free Borland C version 5.5 compiler.

Type:

```
makelib -se -lcc
```

to compile the C code with elj-win32's lcc C compiler.

If you have both the Borland C compiler and lcc installed, make sure the `make.exe` in your path is the correct one!

1.2.3 Library naming conventions

The name of this library starts with `libeposix`. On Unix the name of the Eiffel vendor is appended, so `libeposix_se.a` is the library for SmallEiffel. On Windows systems the name of the Eiffel vendor and the C compiler are appended. On Windows different C compilers have incompatible libraries, so they need to be distinguishes. On Windows the e-POSIX library for ISE Eiffel compiled with the Microsoft Visual C compiler is called `libeposix_ise_msc.lib`.

The vendor names are derived from the names the Gobo Eiffel package uses, i.e. the `GOBO_EIFFEL` environment variable.

In this chapter:

- ***Using*** `library.xace`
- ***Vendor specific notes***
- ***Platform specific notes***

2 Using e-POSIX

2.1 *Using* `library.xace`

Since Gobo 3.0 Eiffel library writes have a new great tool at their dispose: `gexace`. Eiffel library writers have to write and maintain just a single file, `library.xace`. You can this file file in the e-POSIX `src` subdirectory.

Typically, a `library.xace` is included in a `system.xace`. A typical example, including all required Gobo files, is:

```
<?xml version="1.0"?>

<system name="eposix_test">
	<description>
		system:      "getest for eposix"
		author:      "Berend de Boer [berend@pobox.com]"
		copyright:   "Copyright (c) 2002, Berend de Boer"
		license:     "Eiffel Forum Freeware License v1 (see forum.txt)"
		date:        "$Date: $"
		revision:    "$Revision: $"
	</description>

	<root class="EPOSIX_TEST" creation="make"/>

	<option unless="${DEBUG}">
		<option name="assertion" value="all"/>
		<option name="garbage_collector" value="none"/>
		<option name="finalize" value="true" unless="${GOBO_EIFFEL}=ve"/>
	</option>
	<option if="${DEBUG}">
		<option name="assertion" value="all"/>
		<option name="garbage_collector" value="internal"/>
		<option name="finalize" value="false"/>
	</option>

	<option name="linker" value="microsoft" if="${GOBO_EIFFEL}=ve"/>

  <cluster name="testgen" location="TESTGEN"/>
  <cluster name="test" location="${EPOSIX}/test_suite/abstract/system"/>

  <mount location="${EPOSIX}/src/library.xace"/>
	<mount location="${GOBO}/library/test/library.xace"/>
```

```
        <mount location="${GOBO}/library/lexical/library.xace"/>
        <mount location="${GOBO}/library/kernel/library.xace"/>
        <mount location="${GOBO}/library/structure/library.xace"/>
        <mount location="${GOBO}/library/parse/library.xace"/>
        <mount location="${GOBO}/library/utility/library.xace"/>
        <mount location="${GOBO}/library/pattern/library.xace"/>
        <mount location="${GOBO}/library/kernel.xace"/>
</system>
```

2.2 Vendor specific notes

2.2.1 ISE Eiffel

e-POSIX supports ISE Eiffel 5.1. e-POSIX has not yet been tested with version 5.2. e-POSIX has been tested under the following conditions:

1. I used Microsoft Windows 2000, Service Pack 2.
2. I used the Borland C 5.5 and Microsoft Visual C++ 6.0 compiler.

2.2.2 SmallEiffel

e-POSIX was developed using SmallEiffel -0.75 on FreeBSD and Linux.

Because SmallEiffel has a tendency to provide lots of non-ELKS routines in its kernel classes, a bad thing in my opinion, I had to write a new *ANY*. My *ANY* renames *GENERAL.remove_file*, so I wouldn't get a conflict with *POSIX_FILE_SYSTEM.remove_file*.

There is no reason for the presence of *GENERAL.remove_file*, I expect this to be removed soon[1], so my *ANY* can be deleted when this has happened.

2.2.3 Visual Eiffel

e-POSIX has been tested with two of ObjectTool's offerings:

1. Their free VisualEiffel 4 for Linux.
2. VisualEiffel Professional 4 for Windows.
3. If you use 4.0, VisualEiffel's STRING class must be patched, see `http://groups.yahoo.com/group/visual-eiffel/message/673`.

Follow these steps to compile with VisualEiffel 4 on Windows:

[1] I wrote that two years ago. . .

1. Make sure the VE_BIN environment variable is set to the Bin directory in the VisualEiffel subdirectory. On my system it is set to `M:/Program Files/ObjectTools/VisualEiffel/Bin`.

2. Create the `libeposix_ve_msc.lib` library using the Microsoft Visual C compiler:

   ```
   makelib -ve -msc
   ```

3. Use `gexace` to generate an `.esd` file.

4. Make sure to set the linker supplier option to Microsoft in your `system.xace` file! So an option like this should be present:

   ```
   <option name="linker" value="microsoft" if="\${GOBO_EIFFEL}=ve"/>
   ```

2.2.4 Halstenbach Eiffel

e-POSIX has not been tested with the Halstenbach Eiffel compiler.

2.3 Platform specific notes

Although e-POSIX should, in principle, run on every platform that supports Standard C or POSIX, it cannot be tested on every platform by me alone. This section gives details about the platforms I've used. The main thing you need to do is to edit e-POSIX's `src/library.xace` to the proper libraries for your platform are linked. The default `src/library.xace` is suited for Linux only.

2.3.1 Linux

The latest version of e-POSIX was tested with RedHat 7.1 with kernel 2.4.19.

2.3.2 FreeBSD

The latest version of e-POSIX was tested with FreeBSD 4.4-STABLE. FreeBSD doesn't support `fdatasync`, so we do a `fsync`there. Cases like that are automatically detected by the `configure` script.

You have to edit `/src/library.xace` to link the proper library for FreeBSD. Look at the comments.

2.3.3 Cygwin

The latest version of e-POSIX was tested with Cygwin 1.3.x. Some remarks:

1. Locking doesn't seem to be supported.

2. fifo's (`mkfifo`) are not supported.

3. No support for `fdatasync`, so we do a `fsync`there.

2.3.4 BeOS

The latest version of e-POSIX was tested with BeOS 5.03. BeOS has a nice POSIX compatibility layer. Some remarks:

1. Locking doesn't seem to be supported.
2. fifo's (`mkfifo`) are not supported.
3. Hard links are not supported, only symbolic links.
4. No support for `fdatasync`, so we do a `fsync`there.
5. BeOS does not treat sockets as file descriptors, so e-POSIX's *SUS_TCP_SOCKET* does not work. Perhaps when I've added `recv`and `sendmsg`and such it will be usuable.

2.3.5 QNX

The latest version of e-POSIX was tested with QNX ?.

2.3.6 Win32

The latest version of e-POSIX was tested with Windows 2000, Service Pack 2. On Win32, Standard C is fully supported. With e-POSIX's abstract layer, parts of POSIX and the Single Unix Specification are also supported. Support isn't as extensive as using the Cygwin tools.

In this chapter:

3 Design notes

3.1 Why an entire reimplementation?

One might wonder why I reimplemented the entire Standard C and POSIX library when most vendors also have classes that deal with files, the file system, signals and such. Unfortunately, these classes are nor complete nor very portable between vendors. For someone who wants to compile against all the major vendors —and there are good reasons to do this— there is currently no portable solution. That's why many portable Eiffel programs more or less contain the same code again and again. There are some attempts to write more portable libraries, for example the **Unix File/Directory Handling Cluster** by Friedrich Dominicus, but they also are not complete nor is the implementation satisfactory. For example they usually have much logic at the C level. I wanted only C glue code: all intelligence should be in the Eiffel code.

Another attempt is done by the Gobo cluster: it attempts to provide users with a set of classes that work accross all Eiffel vendors by using only the native facilities offered by each implementation. This approach has the advantage that no C compilation is necessary. The disadvantages are:

1. The contract for these classes is probably not specifiable: for which platforms and which assumptions are the contracts valid? Are these contracts the same in all implementations?

2. It is incomplete, i.e. it doesn't cover most of the POSIX routines.

That's why I started to make the entire Standard C and POSIX routines available to Eiffel programmers. All these routines are nicely wrapped in classes. I spend a lot of time designing and refactoring these, comments and improvements about its structure are very appreciated.

The advantage of making POSIX available to Eiffel programmers is that someone doesn't need to think about creating a set of portable file and directory classes that work on every known operating system. POSIX is available on many platforms and for other systems there either is an emulation or a POSIX mapping available. It's better to reuse that, instead of reinventing work that took years to complete.

3.2 Goals and guidelines

The goals and guidelines for this library were:

1. A complete Standard C implementation for those who didn't have access to POSIX routines.

2. A complete POSIX implementation.

3. Do the job in such a way that it will become the official Eiffel POSIX mapping.
4. All classes should satisfy the demands posed by the query–command separation principle.
5. The native Standard C and POSIX routines should be available to those who don't want to go through a certain class layer.
6. The names in use in the POSIX world like file descriptor or memory map are used as class names. This should make it easy to find a class if one knows the POSIX name.
7. If a command fails, an exception code is raised. This differs from the POSIX routines where one is expected to test for error and query the `errno`variable. The only exception is `unlink`: when the file does not exist, no exception is raised.
8. POSIX assumptions should be made explicit. For Eiffel this means specifying explicit pre– and postconditions.
9. Use of constants to influence the way a method should be avoided by providing clearly named methods. So instead of passing a constants to the *POSIX_FILE.open* function to open a file read–only, one can also call *open_read*.
10. Attempt to create non-deferred class that refer to an entity that exists in the POSIX world. Creation of an object is binding to that entity, or creation of that entity.
11. Names should be clear, and Eiffel–like. They should not differ in just one character. POSIX names are also made available to ease use of this library for programmers that know POSIX well.

3.3 Class structure

e-POSIX makes available all the Standard C and POSIX headers in classes like *CAPI_STDIO* and *PAPI_UNISTD*. More details about the header translation are in **chapter 12**.

However, making the plain C API available is not a very interesting addition to an Eiffel programmer's toolkit. Therefore, this library's second attempt was to make an effective OO–wrapper, while making a careful distinction between what is available in the Standard C and what is available in POSIX. This distinction is reflected in e-POSIX's directory structure, see **figure 3.1**.

The raw Standard C API is available in `src/capi`, the OO–wrapper is available in `src/standardc`. The raw POSIX API is available in `src/papi`, the OO–wrapper is available in `src/posix`.

Every Standard C and POSIX wrapper is derived from a common root, see also **figure 3.2**:

1. If a class builds upon facilities available on Standard C, its name starts with the prefix STDC_ and it inherits from *STDC_BASE*.
2. If a class builds upon facilities available in POSIX, its name starts with the prefix POSIX_ and it inherits from *POSIX_BASE*.
3. If a class builds upon facilities available in the Single Unix Specification, its name starts with the prefix SUS_ and it inherits from *SUS_BASE*. The support for the Single Unix Specification is not yet complete, but is continually enhanced.
4. Because we live in a world dominated by Microsoft Windows, and Microsoft Windows does not do POSIX, this would mean that many users only could use e-POSIX's Standard C facilities.

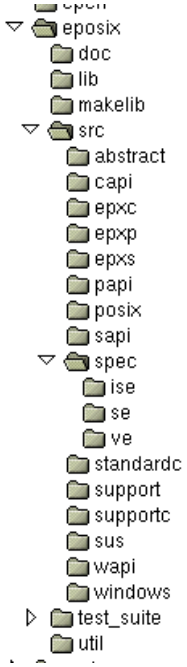

Figure 3.1 e-POSIX directory structure

These facilities are extremely limiting, for example there is no change directory command in Standard C. Therefore e-POSIX makes available an abstraction layer that covers routines that have an equivalent in POSIX and the Single Unix Specification. These classes start with the name EPX_. They always inherit from classes starting with ABSTRACT_. These abstract classes implement the common code. See **chapter 9.2** for more details.

Note that by using Cygwin you have a full POSIX emulation layer on Windows. In that specific environment you can use e-POSIX's entire POSIX and Single Unix Specification layer.

The wrapper classes should be fully command–query separated and use clear names. Often the POSIX name, if applicable, is also made available as an alias. If this is a good thing, I'm not sure. I hope it facilitates working with the wrapper classes if you already know POSIX.

Besides these directories, e-POSIX provides a number of extensions to the pure Standard C or POSIX routines. These can be found in the subdirectories that start with `src/epx`. A single letter indicates if the classes only built upon routines available in Standard C or POSIX:

1. `epxc`: Standard C based extensions like URI resolving, a MIME parser and XML generation.

2. `epxs`: Single Unix Specification based extension like an HTTP client.

3.4 Clients of this library

For client classes, two important classes are *STDC_CONSTANTS* and *POSIX_CONSTANTS*, see **figure 3.3**. The wrapper classes tend to avoid having routines whose behavior drastically depends

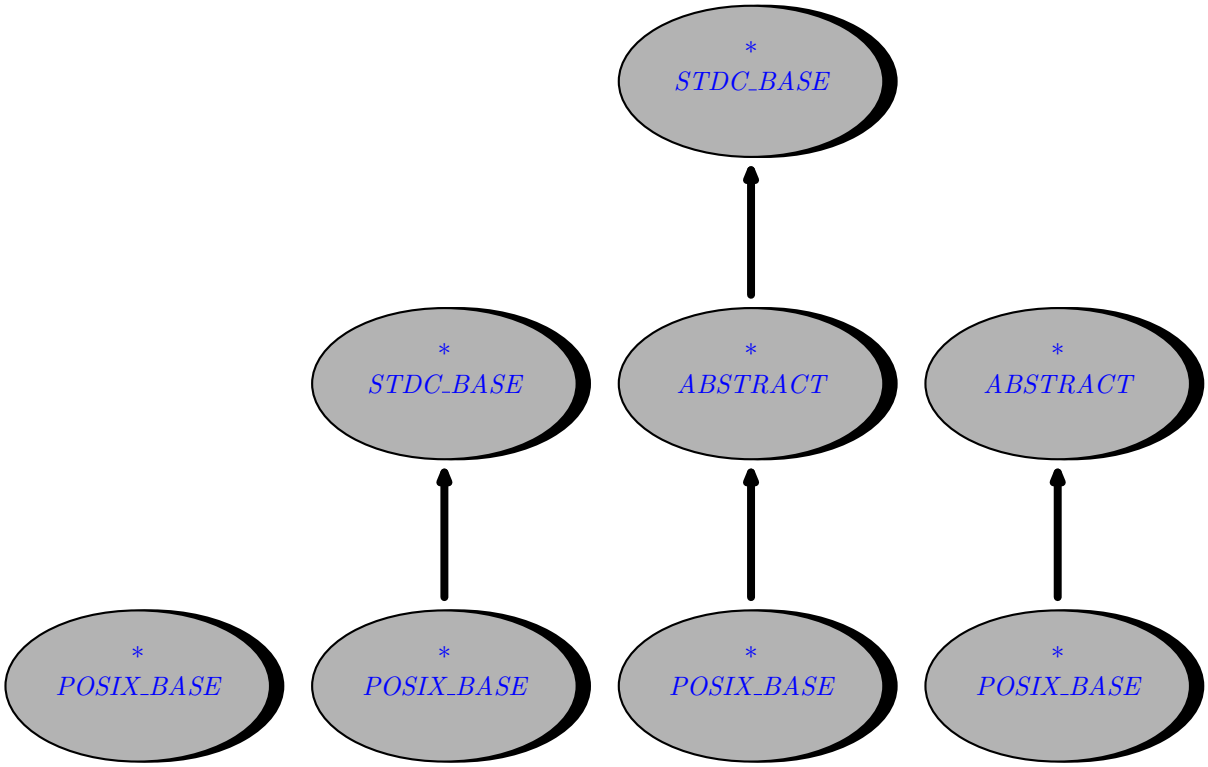

Figure 3.2 Inheritance structure

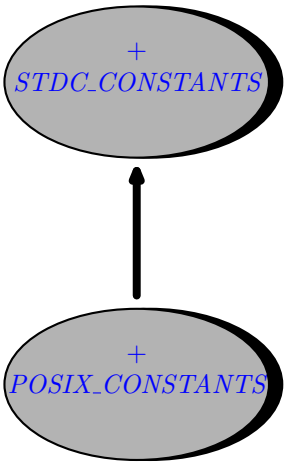

Figure 3.3 Standard C and POSIX constants

on passed constants. But if you need to use constants, your client class can just inherit from these classes and every Standard C and POSIX constant is available.

3.5 Forking

Implementing forking posed some interesting challenges. I started with the basic idea that every process has a pid:

```
class PROCESS

feature

    pid: INTEGER

end
```

I wanted to be able to write two kinds of forking. The first one is forking a child as in:

```
class PARENT

inherit

    POSIX_CURRENT_PROCESS

feature

    make is
        local
            child: POSIX_CHILD_PROCESS
        do
            print ("My pid: ")
            print (pid)
            print ("%N")
            fork (child)
            print ("child's pid: ")
            print (child.pid)
            print ("%N")
            child.wait_for (True)
        end

end
```

However, I also wanted to fork myself, because that basically is what forking is!

```
class PARENT

inherit

    POSIX_CURRENT_PROCESS

    POSIX_CHILD_PROCESS

feature
```

```
    make is
        do
            fork (Current)
            wait
        end

    execute is
        do
            -- forked code
        end

end
```

The above code gives a name clash, because *POSIX_CURRENT_PROCESS.pid* is a call to the POSIX routine `getpid`, while the child's pid is a variable, which gets a variable after forking. You can solve this name clash yourself, but it is most easy to inherit from *POSIX_FORK_ROOT*, a clash which has solved this clash already.

If you fork a child, you must wait for it. For a child process, you can use *POSIX_CHILD.wait_for*, if you fork yourself, you must use *POSIX_CURRENT_PROCESS.wait*. The variable *waited_child_pid* will be set with the pid of the child process that *wait* waited for.

3.6 Books

Books that have been helpful during the development of e-POSIX where [43], see the biography section at **page 193**.

In this chapter:

- ***Working with files***
- ***Working with file descriptors***
- ***Working with the file system***
- ***Executing a child command***
- ***Current time***
- ***Accessing environment variables***
- ***Allocating memory***
- ***Redirecting stderr to stdout***

4 Basic Posix examples

Instead of describing every class and every feature, I decided to show short and simple examples of common ways to use the Posix library features. If you don't have Posix available, you can try to replace the POSIX prefix by STDC. Most of the time the POSIX classes are based on the STDC classes, see **chapter 7**.

4.1 Working with files

The basic class for working with files, or streams as they are also called, is *POSIX_FILE*. There are two kinds of files: *POSIX_TEXT_FILE* and *POSIX_BINARY_FILE*. According to the POSIX standard, there is no distinction between binary and text files. But on certain systems you must use POSIX programs through an emulation layer. For example, on Windows Cygwin is a well–known POSIX emulator. To maintain compatibility with other Windows programs, Cygwin distinguishes between text and binary files. If you use Cygwin to compile your POSIX programs, this distinction is therefore still important.

The first example shows how to open a text file, see also the corresponding BON diagram in **figure 4.1**.

```
class EX_FILE1

creation

  make

feature

  make is
    local
      file: POSIX_TEXT_FILE
    do
      create file.open_read ("/etc/group")
      from
        file.read_string (256)
      until
        file.eof
```

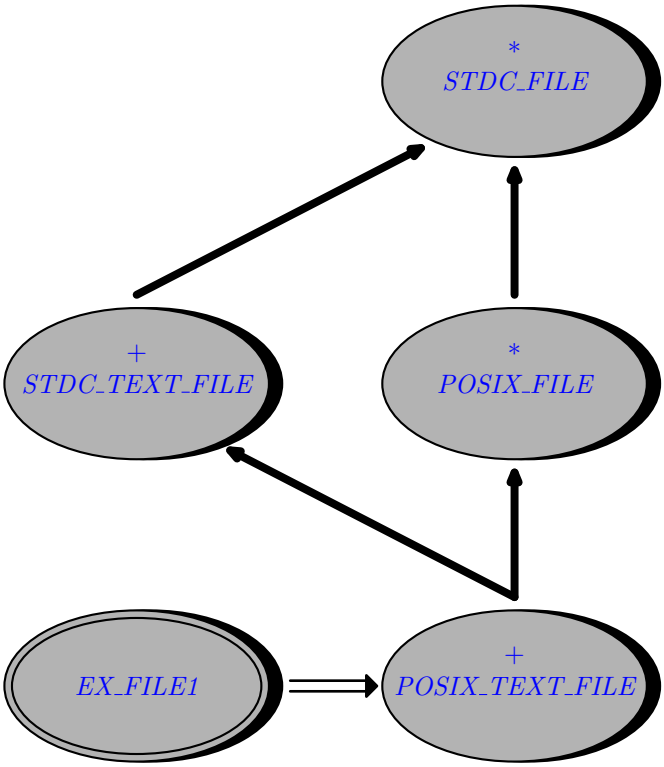

Figure 4.1 BON diagram of opening a text file.

```
    loop
      print (file.last_string)
      file.read_string (256)
    end
    file.close
  end

end
```

It simply opens a file for reading and prints every line in it. Note that you have to specify the maximum number of characters you are prepared to read. The minimum characters read are 256, but perhaps you want to be able to read text files consisting of 1024 characters per line.

Every line that is read includes the end–of–line character if one was present. This is unlike Pascal for example, but more like Perl. e-POSIX provides the feature *POSIX_TEXT_FILE.chop* which removes the last character of *last_string* if and only if it is an end–of–line character. And that is unlike Perl, which removes any character. With e-POSIX it is not necesary to test for the end–of–line characters if you just want to remove it in case one is present.

At the end, the file is closed. You don't need to explicitly close a file as it will be closed when your object is garbaged collected. But I think it's a good thing not to rely or depend on this, but to close your external resources as soon as you're done using them. For example many systems have easily reached limits on the number of files a process can have open.

Reading binary files is almost the same loop, only you read it in chunks:

```
class EX_FILE2

creation

    make

feature

    chunk_size: INTEGER is 512

    make is
        local
            file: POSIX_BINARY_FILE
            buffer: POSIX_BUFFER
        do
            create file.open_read ("/bin/sh")
            create buffer.allocate (chunk_size)
            from
                file.read_buffer (buffer, 0, chunk_size)
            until
                file.eof
            loop
                file.read_buffer (buffer, 0, chunk_size)
            end
            file.close
        end

end
```

This example uses a more safe version of buffer reading, *POSIX_FILE.read_buffer*. There is an untyped variant *POSIX_FILE.read* which accepts a pure pointer. There is no need to mention that you need to watch buffer overflows carefully with this last one!

Correctly looping through files, takes care. For example the following loop also works, but is less correct:

```
class EX_WRONG1

creation

    make

feature

    make is
        local
            file: POSIX_TEXT_FILE
        do
            create file.open_read ("/etc/group")
```

```
            from
            until
                file.eof
            loop
                file.read_string (256)
                print (file.last_string)
            end
            file.close
        end

end
```

After *POSIX_TEXT_FILE.read_string*, *eof* might be True. Because the string is empty in that case, nothing will be printed. You will make an unnecessary extra loop. The correctly coded variant is:

```
class EX_WRONG2

creation

    make

feature

    make is
        local
            file: POSIX_TEXT_FILE
        do
            create file.open_read ("/etc/group")
            from
            until
                file.eof
            loop
                file.read_string (256)
                if not file.eof then
                    print (file.last_string)
                end
            end
            file.close
        end

end
```

I myself prefer the first example, as the check is only in the **until** part, and not repeated in the loop.

The following examples shows how a binary file is created and a string is written to it.

```
class EX_FILE3

inherit

    POSIX_FILE_SYSTEM
```

```
creation

    make

feature

    make is
        local
            file: POSIX_BINARY_FILE
        do
            create file.create_write (expand_path ("$HOME/myfile.tmp"))
            file.write_string ("hello world.%N")
            file.close
        end

end
```

Depending on the platform you are running a backslash is turned into a slash or vice versa.

This example also demonstrates how path names —file and directory names— can be expanded: if you call *POSIX_FILE_SYSTEM.expand_path*, any environment variables in the path are expanded. Backslashes and slashes are always translated, but environment variable expansion has to be done explicitly.

You can move the file pointer with two different methods: *POSIX_FILE.seek* and *set_position*. The *seek* works with files up to 2 GB, *set_position* has no such limits. Use *tell* to get a position that can be passed to *seek*. Use *get_position* to get a position that can be passed to *set_position*.

```
class EX_FILE5

creation

    make

feature

    make is
        local
            file: POSIX_BINARY_FILE
            pos1: INTEGER
            pos2: STDC_FILE_POSITION
        do
            create file.create_read_write ("test.bin")
            file.write_string ("one")
            pos1 := file.tell
            pos2 := file.get_position
            file.write_string ("two")
            file.seek (pos1)
            -- or file.set_position (pos2)
            file.read_string (3)
```

```
            if not file.last_string.is_equal ("two") then
                print ("unexpected read.%N")
            end
            file.close
        end

end
```

4.2 Working with file descriptors

The file descriptors classes are quite equal to the file classes. The following example opens a file using *POSIX_FILE_DESCRIPTOR* and reads the first 64 bytes.

```
class EX_FD1

creation

    make

feature

    make is
        local
            fd: POSIX_FILE_DESCRIPTOR
        do
            create fd.open_read ("/etc/group")
            fd.read_string (64)
            print (fd.last_string)
            fd.close
        end

end
```

Unlike *POSIX_TEXT_FILE*, there is no easy way to detect end of line and end of file conditions. However, a file descriptor can easily be turned into a file as the following example demonstrates.

```
class EX_FD2

creation

    make

feature

    make is
        local
            fd: POSIX_FILE_DESCRIPTOR
            file: POSIX_TEXT_FILE
```

```
        do
            create fd.open_read ("/etc/group")
            create file.make_from_file_descriptor (fd, "r")
            from
                file.read_string (256)
            until
                file.eof
            loop
                print (file.last_string)
                file.read_string (256)
            end
            file.close
            fd.close
        end

end
```

A file descriptor can also be used to lock, unlock or test for locks on a given file as the following example demonstrates. See also the accompanying BON diagram in **figure 4.2**.

```
class EX_FD4

creation

    make

feature

    make is
        local
            some_lock,
            lock: POSIX_LOCK
            fd: POSIX_FILE_DESCRIPTOR
        do
            create fd.create_read_write ("test.tmp")
            fd.write_string ("Test")

            create lock.make
            lock.set_allow_read
            lock.set_start (2)
            lock.set_length (1)
            some_lock := fd.get_lock (lock)
            if some_lock /= Void then
                print ("There is already a lock?%N")
            end

            -- create exclusive lock
            lock.set_allow_none
```

```
            lock.set_start (0)
            lock.set_length (4)
            fd.set_lock (lock)

            fd.close
        end

end
```

POSIX_FILE_DESCRIPTOR.get_lock is command–query separated, that is why it returns a new lock when queried and there is a lock. If there is no lock *get_lock* returns Void. The passed lock is not modified.

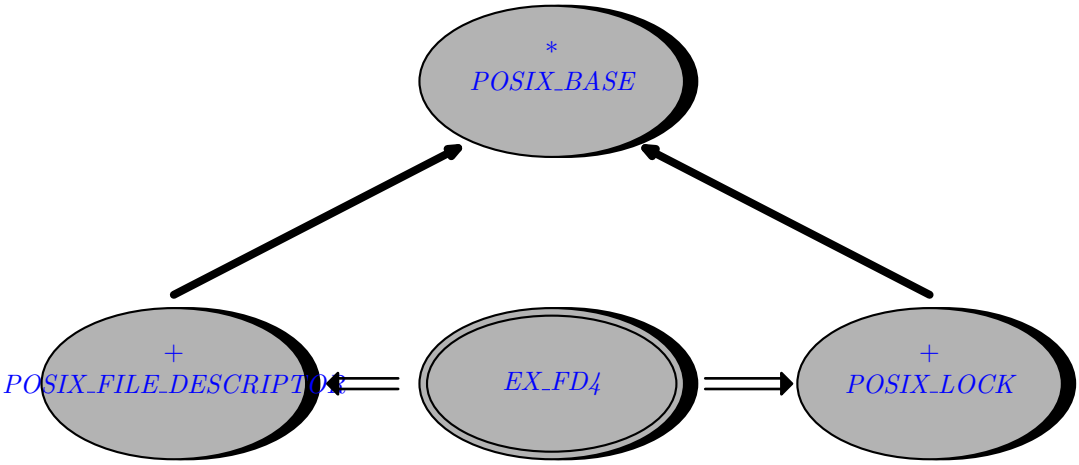

Figure 4.2 BON diagram of locking a portion of a file.

A file descriptor also gives you access to the attached terminal, if any. The following example demonstrates how to read a password without the password appearing on the screen.

```
class EX_FD3

inherit

    POSIX_CURRENT_PROCESS

creation

    make

feature

    make is
        do
            print ("Password: ")
            stdout.flush
```

```
        -- turn off echo
        fd_stdin.terminal.set_echo_input (False)
        fd_stdin.terminal.apply_flush

        -- read password
        fd_stdin.read_string (256)

        -- turn echo back on
        fd_stdin.terminal.set_echo_input (True)
        fd_stdin.terminal.apply_now

        print ("%NYour password was: ")
        print (fd_stdin.last_string)
    end

end
```

4.3 Working with the file system

POSIX defines many commands to navigate a file system. They're made available by the *POSIX_FILE_SYSTEM*. The following example navigates to the user's home directory, create a directory and removes it.

```
class EX_DIR1

inherit

    POSIX_FILE_SYSTEM

creation

    make

feature

    make is
        do
            change_directory (expand_path ("~"))
            make_directory ("qqtest.xyz.tmp")
            remove_directory ("qqtest.xyz.tmp")
        end

end
```

To get access to the file system, inheriting from the *POSIX_FILE_SYSTEM* class is easiest.

There are also lots of functions to test for existence, readability or writability of files. Use *is_modifiable* to test if a file is readable and writable.

```
class EX_DIR2

inherit

    POSIX_FILE_SYSTEM

creation

    make

feature

    make is
        local
            perm: POSIX_PERMISSIONS
        do
            print_info (is_existing ("/tmp"), "existing")
            print_info (is_executable ("/bin/ls"), "executable")
            print_info (is_readable ("/etc/passwd"), "readable")
            print_info (is_writable ("/etc/passwd"), "writable")
            print_info (is_modifiable ("/etc/passwd"), "readable and writable")

            perm := permissions("/etc/passwd")

            if perm.allow_group_read then
                print ("Group is allowed to read /etc/passwd.%N")
            else
                print ("Group is not allowed to read /etc/passwd.%N")
            end

            if perm.allow_anyone_read_write then
                print ("Anyone is allowed to read file.tmp.%N")
            else
                print ("Anyone is not allowed to read file.tmp.%N")
            end

        end

    print_info (ok: BOOLEAN; what: STRING) is
        do
            print ("is_")
            print (what)
            print (" returned ")
            print (ok)
            print (".%N")
        end
```

```
end
```

Be aware that *POSIX_FILE_SYSTEM.is_readable* uses the real user and group IDs instead of the effective ones.

As can be seen in the above example, one can test for the permissions of a file using the *POSIX_PERMISSIONS* class. A new permissions class is created for every *POSIX_FILE_SYSTEM.permissions* call, so it is best to cache this object. If the permissions change on the file system, this class does not reflect reality anymore, because it caches the permissions. Use *POSIX_PERMISSIONS.refresh* to update the contents. Use *set_allow_group_write*, *set_allow_anyone_read* and such to set permissions.

e-POSIX also gives you access to the `stat`function using the *POSIX_STATUS* class.

```
class EX_DIR4

inherit

    POSIX_FILE_SYSTEM

creation

    make

feature

    make is
        local
            stat: POSIX_STATUS
        do
            stat := status ("/etc/passwd")
            print ("size: ")
            print (stat.size.out)
            print (".%N")
            print ("uid: ")
            print (stat.permissions.uid)
            print (".%N")
        end

end
```

The *POSIX_STAT*, and through it *POSIX_PERMISSIONS*, are also returned by *POSIX_FILE_DESCRIPTOR.status*.

Browsing a directory can be done by allocated a *POSIX_DIRECTORY* class through the *POSIX_FILE_SYSTEM.browse_directory* feature:

```
class EX_DIR3

inherit

    POSIX_FILE_SYSTEM
```

```
creation

    make

feature

    make is
        local
            dir: POSIX_DIRECTORY
        do
            from
                dir := browse_directory (".")
                dir.start
            until
                dir.exhausted
            loop
                print (dir.item)
                print ("%N")
                dir.forth
            end
            dir.close
        end

end
```

As can be seen, *POSIX_DIRECTORY* follows EiffelBase conventions.

When browsing a directory, all entries in that directory are returned. You might want to be interested only in certain files. e-POSIX has the ability to define arbitrary filters. Standard e-POSIX comes with an extension filter that only shows files with a certain extension:

```
class EX_DIR6

inherit

    POSIX_FILE_SYSTEM

creation

    make

feature

    make is
        local
            dir: POSIX_DIRECTORY
        do
            from
                dir := browse_directory (".")
```

```
            dir.set_extension_filter (".e")
            dir.start
        until
            dir.exhausted
        loop
            print (dir.item)
            print ("%N")
            dir.forth
        end
        dir.close
    end

end
```

4.4 Executing a child command

Any command line can be executed by using the *POSIX_SHELL_COMMAND* class. Just pass a command line and *execute* it.

```
class EX_CMD

creation

    make

feature

    make is
        local
            command: POSIX_SHELL_COMMAND
        do
            create command.make ("/bin/ls *")
            command.execute
            print ("Exit code: ")
            print (command.exit_code)
            print ("%N")
        end

end
```

Often one wants to redirect the output of the program that is being executed. For such cases use the *POSIX_EXEC_PROCESS* class.

```
class EX_EXEC1

inherit

  POSIX_CURRENT_PROCESS
```

```
creation

 make

feature

 make is
  local
   ls: POSIX_EXEC_PROCESS
  do
   -- list contents of current directory
   create ls.make_capture_output ("ls", <<"-1", ".">>)
   ls.execute
   print ("ls pid: ")
   print (ls.pid)
   print ("%N")
   from
    ls.stdout.read_string (512)
   until
    ls.stdout.eof
   loop
    print (ls.stdout.last_string)
    ls.stdout.read_string (512)
   end

   -- close captured io
   ls.stdout.close

   -- wait for process
   ls.wait_for (True)
  end

end
```

Besides capturing output, you can also capture the standard input and standard error of the executed process.

It is important to wait for the child that has been executed at some point in time, just like any POSIX would have to do. If you do not wait for a child process, memory in the kernel is not released and eventually you would run out of processes. Also, after the *POSIX_EXEC_PROCESS.wait_for* command, the exit code of the process becomes available.

4.5 Current time

e-POSIXhas a very complete class to work with times. A time can be set from the current time by using *POSIX_TIME.make_from_now*. Before a time can be printed, it needs to be converted to either local time or UTC. Do this by calling *to_local* or *to_utc*. Date and times can be printed

using features as *default_format*, *local_date_string*, *local_time_string* or a custom format through *format*.

```
class EX_TIME1

creation

    make

feature

    make is
        local
            time1,
            time2: POSIX_TIME
        do
            create time1.make_from_now
            time1.to_local
            print_time (time1)
            time1.to_utc
            print_time (time1)
            create time2.make_time (0, 0, 0)
            print_time (time2)
            create time2.make_date_time (1970, 10, 31, 6, 55, 0)
            time2.to_utc
            print_time (time2)

            if time2 < time1 then
                print ("time2 is less than time1 as expected.%N")
            else
                print ("!! time2 is not less than time1.%N")
            end
        end

    print_time (time: POSIX_TIME) is
        do
            print ("Date: ")
            print (time.year)
            print ("-")
            print (time.month)
            print ("-")
            print (time.day)
            print (" ")
            print (time.hour)
            print (":")
            print (time.minute)
            print (":")
            print (time.second)
```

```
            print ("%N")
            print ("Weekday: ")
            print (time.weekday)
            print ("%N")
            print ("default string: ")
            print (time.default_format)
            print ("%N")
        end

end
```

4.6 Accessing environment variables

With the class *POSIX_ENV_VAR*, the contents of environment variables can be queried.

```
class EX_ENV1

creation

    make

feature

    make is
        local
            env: POSIX_ENV_VAR
        do
            create env.make ("HOME")
            print (env.value)
            print ("%N")
        end

end
```

Unfortunately, it is not possible in POSIX to set an environment variable. This is possible with the Single Unix Specification classes, see **section 6.1**.

4.7 Allocating memory

Allocating dynamic memory is very useful, but not portably available for Eiffel programmers. With *POSIX_BUFFER* memory can be allocated, read and written to.

```
class EX_MEM

creation

    make
```

```
feature

    make is
        local
            mem: POSIX_BUFFER
            byte: INTEGER
        do
            create mem.allocate (256)
            mem.poke_uint8 (2, 57)
            byte := mem.peek_uint8 (2)
            mem.resize (512)
            mem.deallocate
        end

end
```

For more information about the dynamic memory class, see **section 7.1**.

4.8 Redirecting stderr to stdout

If you want to redirect all output written by your program or any child you spawn to stdout, you can use the *POSIX_FILE_DESCRIPTOR.make_as_duplicate* call:

```
class EX_REDIRECT1

inherit

    POSIX_CURRENT_PROCESS

creation

    make

feature

    make is
        do
            -- flush stream buffers, else output may be in wrong order
            stdout.flush
            stderr.flush

            fd_stderr.make_as_duplicate (fd_stdout)
            -- all output written to stderr goes to stdout now
        end

end
```

It's a good idea to call this at the beginning of your program, before you have written anything to stderr or stdout. If you do that, you don't have to flush the stream buffers.

In this chapter:

- *Catching a signal*
- *General wait for child handler*
- *Forking a child process*
- *Creating a daemon*
- *Asynchronous I/O*
- *Talking to your modem*
- *Using shared memory*
- *More examples*

5 Advanced Posix examples

5.1 Catching a signal

Every class can become a signal handler by inheriting from *POSIX_SIGNAL_HANDLER*. Implement the *signalled* method as that is the function that is called when the signal occurs. Use *POSIX_SIGNAL.set_handler* to make your class a signal handler and call *apply* to start receiving signals when they occur.

The following examples demonstrates a program that can be safely interrupted by pressing Ctrl+C:

```
class EX_SIGNAL1

inherit

    POSIX_CURRENT_PROCESS

    POSIX_CONSTANTS

    POSIX_SIGNAL_HANDLER

creation

    make

feature

    handled: BOOLEAN

    make is
        local
            signal: POSIX_SIGNAL
        do
            create signal.make (SIGINT)
            signal.set_handler (Current)
            signal.apply
```

```
            print ("Wait 30s or press Ctrl+C.%N")
            sleep (30)
            if handled then
                print ("Ctrl+C pressed.%N")
            else
                print ("Ctrl+C not pressed.%N")
            end
        end

    signalled (signal_value: INTEGER) is
        do
            handled := True
        end

end
```

All precautions and warnings when handling signals in C apply equally well in Eiffel of course. While in a signal handler, the signal will not be delivered again. Call *STDC_SIGNAL_HANDLER.reestablish* to make your signal handler interruptable.

You can write a single signal handler, that handles multiple signals. This makes it possible to have signal handling code in just one place. Create a class that inherits from *POSIX_SIGNAL_HANDLER*. Pass this class to the *POSIX_SIGNAL.set_handler* for every signal you want to catch. The signal value is passed as parameter to *POSIX_SIGNAL_HANDLER.signalled*, so you can write an **inspect** statement based on the value.

5.2 General wait for child handler

If you do not want to wait for every child process explicitly, you can write a simple SIGCHLD handler that just does a wait (I found this idea in [43]):

```
class EX_SIGNAL2

inherit

    POSIX_CURRENT_PROCESS

    POSIX_CONSTANTS

    POSIX_SIGNAL_HANDLER

creation

    make

feature

    make is
        local
```

```
            signal: POSIX_SIGNAL
        do
            create signal.make (SIGCHLD)
            signal.set_handler (Current)
            signal.apply

            -- spawn child processes here
            -- you dont have to wait for them
        end

    signalled (signal_value: INTEGER) is
        do
            wait
        end

end
```

In Unix 98 you should be able to set the ignore handler for this signal. In pure POSIX systems the behaviour of the ignore handler is unspecified.

5.3 Forking a child process

Forking is very easy with this Eiffel POSIX implementation. The steps:

1. Write a child by inheriting from *POSIX_FORK_ROOT* and implementing its *execute* method.
2. The class that will do the forking, should inherit from *POSIX_CURRENT_PROCESS*.
3. Pass the child to the inherited feature *POSIX_CURRENT_PROCESS.fork* and the forking has begun.

The following class shows the process that forks the child.

```
class

    EX_FORK1

inherit

    POSIX_CURRENT_PROCESS

    POSIX_FILE_SYSTEM

creation

    make

feature
```

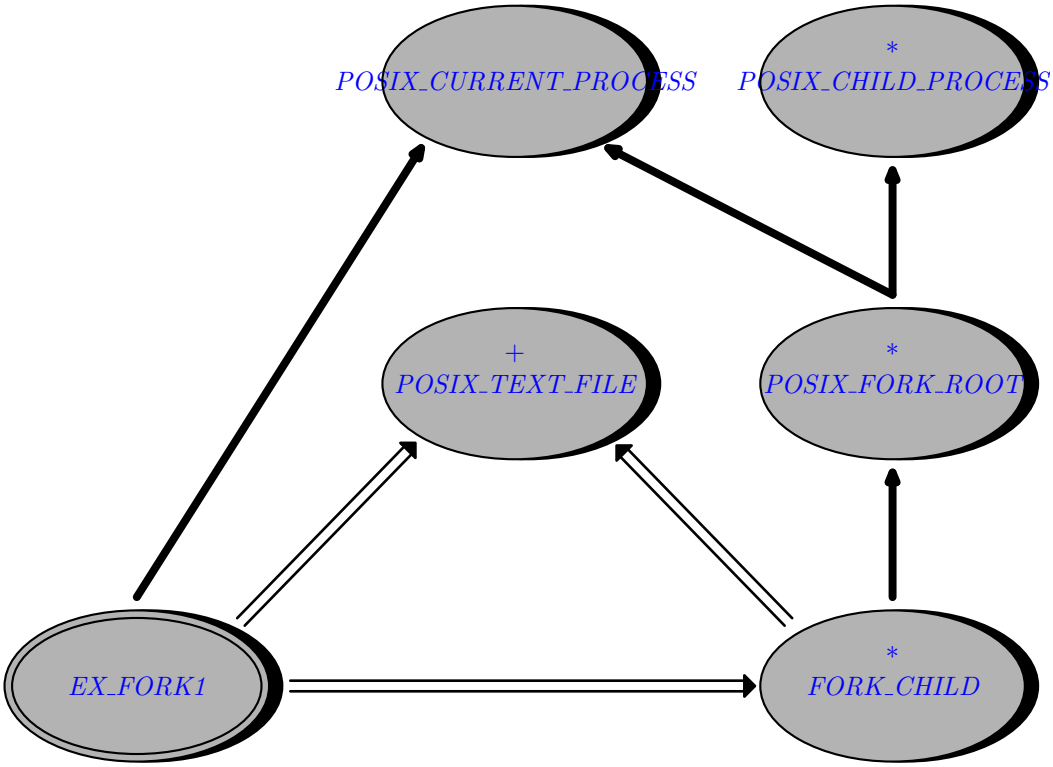

Figure 5.1 BON diagram of forking a child process.

```
make is
    local
        reader: POSIX_TEXT_FILE
        stop_sign: BOOLEAN
        child: FORK_CHILD
    do
        -- necessary for SmallEiffel before -0.75 beta 7
        ignore_child_stop_signal

        unlink ("berend.tmp")
        create_fifo ("berend.tmp", S_IRUSR + S_IWUSR)
        create child
        fork (child)

        -- we will now block until file is opened for writing
        create reader.open_read ("berend.tmp")
        from
            stop_sign := False
        until
            stop_sign
        loop
            reader.read_string (128)
```

```
            print (reader.last_string)
            stop_sign := equal(reader.last_string, "stop%N")
        end
        reader.close

        -- now wait for the writer to terminate
        child.wait_for (True)

        unlink ("berend.tmp")
    end

end
```

This class just displays anything that the writer, the child class, writes to the FIFO. When it recognizes stop, the reader stops after waiting for the child it has spawned. Note that this is very important! Wait for any child you have spawned else you might get spurious errors if the process exits and a child has not yet finished.

The following class shows the forked child.

```
class FORK_CHILD

inherit

    POSIX_FORK_ROOT

feature

    execute is
        local
            writer: POSIX_TEXT_FILE
        do
            create writer.open_append ("berend.tmp")
            writer.write_string ("first%N")
            writer.write_string ("stop%N")
            writer.close

            -- we give the reader some time to process these messages
            sleep (10)
        end

end
```

5.4 Creating a daemon

Creating a simple daemon is easy if you inherit from *POSIX_DAEMON*. Implement the *execute* method, and you're done. At run–time, call *detach* to fork off a child. You can call *detach* as many times as you want to spawn daemons.

```
class EX_DAEMON

inherit

 POSIX_DAEMON

 ARGUMENTS

creation

 make

feature -- the parent

 make is
  do
   -- necessary under SmallEiffel
   ignore_child_stop_signal

   if argument_count = 0 then
    print ("Options:%N")
    print ("-d      start daemon%N")
   else
    if equal(argument(1), "-d") then
     detach
     print ("Daemon started.%N")
     print ("Its pid: ")
     print (last_child_pid)
     print ("%N")
    end
   end
  end

feature -- the daemon

 execute is
  do
   -- daemon stays alive for 20 seconds
   sleep (20)
  end

end
```

5.5 Asynchronous I/O

e-POSIX supports the asynchronous i/o features of POSIX. Not all Free Unices seem to support this feature, nor does their support seems to be error free.

Take a look at the following example:

```
class EX_ASYNC1

creation

    make

feature

    make is
        local
            fd: POSIX_FILE_DESCRIPTOR
            request: POSIX_ASYNC_IO_REQUEST
        do
            create fd.create_read_write ("test.tmp")
            create request.make (fd)
            request.set_offset (0)
            request.write_string ("hello world.")
            request.wait_for
            fd.close
        end

end
```

The basic idea is that each asynchronous request is a separate object, modeled by *POSIX_ASYNC-_IO_REQUEST*. You prepare it through calls like *set_buffer*, *set_count* and *set_offset*. You execute the request by calling *read* or *write*.

You can wait for the request to be complete by calling *wait_for*. It should be possible to force open requests to be synchronized to the disk with *synchronize*, but this does give strange results on Linux. So far I haven't got access to a machine that also implements asynchronous i/o to test if my code is correct.

5.6 Talking to your modem

With e-POSIX you can talk to your modem. The implementation contains not all the details to write a full–featured program as minicom, but they will be added upon request.

The following example tries to talk to your modem —which is expected to be at `/dev/modem`— and queries its manufacturer.

```
class EX_MODEM

inherit

    POSIX_CURRENT_PROCESS

creation
```

```
    make

feature

    make is
        local
            modem: POSIX_FILE_DESCRIPTOR
            term: POSIX_TERMIOS
        do
            -- assume there is a /dev/modem device
            create modem.open_read_write ("/dev/modem")
            term := modem.terminal
            term.flush_input
            print ("Input speed: ")
            print (term.speed_to_baud_rate (term.input_speed))
            print ("%N")
            print ("Output speed: ")
            print (term.speed_to_baud_rate (term.output_speed))
            print ("%N")

            term.set_input_speed (B9600)
            term.set_output_speed (B9600)
            term.set_receive (True)
            term.set_echo_input (False)
            term.set_echo_new_line (False)
            term.set_input_control (True)
            term.apply_flush

            -- expect modem to echo commands
            modem.write_string ("AT%N")
            modem.read_string (64)
            print ("Command: ")
            print (modem.last_string)
            modem.read_string (64)
            print ("Response (expect ok): ")
            print (modem.last_string)
            modem.write_string ("ATI0%N")
            modem.read_string (64)
            print ("Command: ")
            print (modem.last_string)
            modem.read_string (64)
            print ("Response: ")
            print (modem.last_string)
            modem.close
        end

end
```

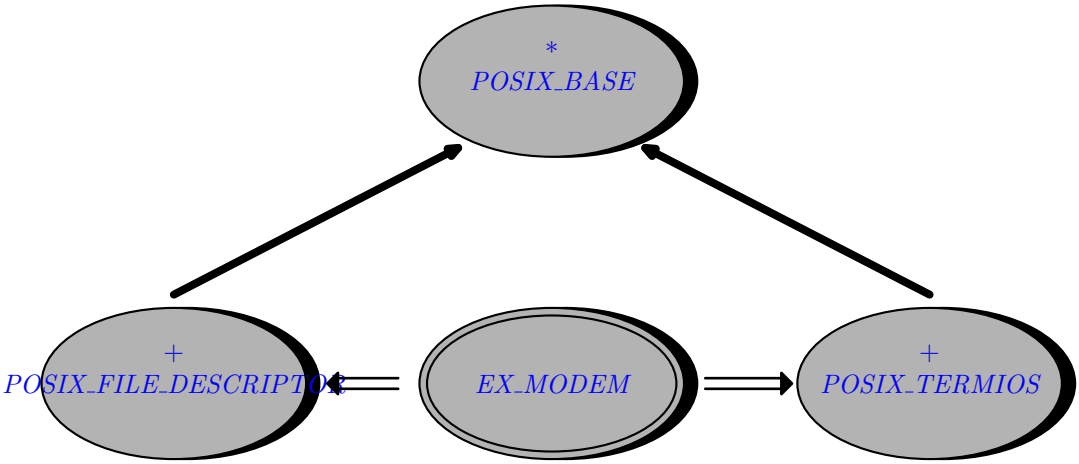

Figure 5.2 BON diagram of talking to a modem.

5.7 Using shared memory

You can use shared memory to exchange data between different processes. It's dependent on your POSIX version if this is supported, so check for this capability explicitly!

```
class EX_SHARED_MEM1

inherit

    POSIX_SYSTEM

    POSIX_CURRENT_PROCESS

    POSIX_FILE_SYSTEM

creation

    make

feature

    make is
        local
            fd: POSIX_SHARED_MEMORY
        do
            if not supports_shared_memory_objects then
                stderr.puts ("Shared memory objects not supported.%N")
                exit_with_failure
            end
```

```
        create fd.create_read_write ("/test.berend")
        fd.write_string ("Hello world.%N")
        fd.close
        unlink_shared_memory_object ("/test.berend")
      end

  end
```

Make sure you always start a shared memory object with a slash. Else the behaviour is undefined or processes might not be able to find your shared memory.

5.8 More examples

If you are looking for more examples, you might take a look at the classes in the `test_suite` directory. These classes should demonstrate and test almost every feature available in the POSIX classes.

In this chapter:

- ***Environment variables***
- ***Logging messages and errors***
- ***Sockets***
- ***HTTP client***

6 Single Unix Specification classes

6.1 Environment variables

Using *SUS_ENV_VAR*set_value it is possible to set environment variables.

6.2 Logging messages and errors

Although POSIX doesn't have logging facilities, the Single Unix Specification does. This specification requires the presence of the `syslogd` daemon for centralizes logging facilities. The following example shows you to write messages to this daemon

```
class EX_SYSLOG

inherit

    SUS_CONSTANTS

    SUS_SYSLOG_ACCESSOR

creation

    make

feature

    make is
        do
            syslog.open ("test", LOG_ODELAY + LOG_PID, LOG_USER)

            syslog.debug_dump ("this is a debug message")
            syslog.info ("this is an informational message")
            syslog.warning ("this is a warning")
            syslog.error ("this is an error message")

            syslog.close
        end
```

```
end
```

Always use the *SUS_SYSLOG_ACCESSOR* to access the syslog wrapper class *SUS_SYSLOG*. *SUS_SYSLOG* is a singleton, it makes no sensen to open a connection to the syslog daemon twice.

6.3 Sockets

e-POSIX currently has initial socket support. It will work only on POSIX systems, support for Windows through the EPX layer will be added in a future release.

The following example demonstrates a simple echo client:

```
class EX_ECHO1

creation

    make

feature

    hello: STRING is "Hello World.%N"

    make is
        local
            host: SUS_HOST
            service: SUS_SERVICE
            echo: SUS_TCP_SOCKET
            sa: SUS_SOCKET_ADDRESS
        do
            -- create host.make_from_name ("localhost")
            create host.make_from_name ("bmach")
            create service.make_from_name ("echo", "tcp")

            create sa.make (host, service)

            create echo.open_by_address (sa)
            echo.write_string (hello)
            echo.read_string (256)
            if not echo.last_string.is_equal (hello) then
                print ("!! got: ")
                print (echo.last_string)
            end
        end

end
```

6.4 HTTP client

The following example demonstrates retrieval of a file through HTTP using the *EPX_HTTP_10_CLIENT* class:

```
class EX_HTTP1

creation

  make

feature

  url: STRING is "http://www.freebsd.org/index.html"

  make is
    local
      uri: EPX_URI
      client: EPX_HTTP_10_CLIENT
    do
      create uri.make (url)
      create client.make (uri.authority) -- www.freebsd.org
      client.get (uri.path) -- /index.html
      client.read_response
      print (client.body.as_string)
    end

end
```

It also demonstrates the use of the *EPX_URI* class to parse an URI into its components.

In this chapter:

- ***Allocating memory***
- ***Accessing environment variables***
- ***Working with streams***
- ***Working with the file system***
- ***Catching a signal***

7 Standard C examples

If you don't have access to a POSIX compatible system, you can use the underlying Standard C classes. Standard C is quite restricted in certain respects: you cannot change directories for example. On the other hand, this library gives you access to all Standard C routines, so you can use what's there and write an extremely portable program.

All Standard C classes start with STDC_. They are:

1. *STDC_TEXT_FILE*: access text files.
2. *STDC_BINARY_FILE*: access binary files.
3. *STC_TEMPORARY_FILE*: create a temporary file, a file that is removed when it is closed or when the program terminates.
4. *STDC_CONSTANTS*: access Standard C constants like error codes and such.
5. *STDC_BUFFER*: allocate dynamic memory.
6. *STDC_ENV_VAR*: access environment variables.
7. *STDC_FILE_SYSTEM*: delete and rename files.
8. *STDC_SHELL_COMMAND*: pass an arbitrary command to the native shell.
9. *STDC_SYSTEM*: access information about the system the program is running on.
10. *STDC_CURRENT_PROCESS*: access to current process related information like its standard input, output and error streams.
11. *STDC_TIME*: access current time. Also can format a given time in various formats.

7.1 Allocating memory

You can dynamically allocate memory with *STDC_BUFFER* which works just like *POSIX_BUFFER*.

class *EX_MEM2*

creation

make

feature

```
    make is
        local
            mem: STDC_BUFFER
            byte: INTEGER
        do
            create mem.allocate_and_clear (128)
            mem.poke_uint8 (2, 57)
            byte := mem.peek_uint8 (2)
            mem.resize (256)
            mem.deallocate
        end

end
```

With the feature *STDC_BUFFER.allocate_and_clear* memory is allocated and cleared to all zeros.

STDC_BUFFER contains many routines to read bytes and strings from the memory it manages like *peek_int16*, *peek_uint16*, or *peek_int32*. It supports reading and writing 16 and 32 bit integers in little and big endian order with routines as *peek_int16_big_endian*, *peek_int16_little_endian*, and *poke_int32_big_endian*.

7.2 Accessing environment variables

Standard C supports reading environment variables with *STDC_ENV_VAR*.

```
class EX_ENV2

creation

    make

feature

    make is
        local
            env: STDC_ENV_VAR
        do
            create env.make ("HOME")
            print (env.value)
            print ("%N")
        end

end
```

7.3 Working with streams

Working with text files is equal to the POSIX classes, only you use the STC prefix.

```
class EX_FILE4
```

```
creation

    make

feature

    make is
        local
            file: STDC_TEXT_FILE
        do
            create file.open_read ("/etc/group")
            from
                file.read_string (256)
            until
                file.eof
            loop
                print (file.last_string)
                file.read_string (256)
            end
            file.close
        end

end
```

Its BON diagram, see **figure 7.1** is therefore quite equal to the POSIX one, see **figure 4.1**.

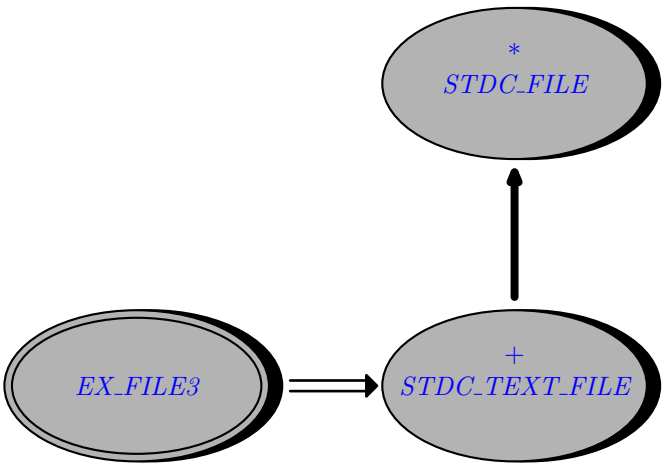

Figure 7.1 BON diagram of opening a Standard C text file.

7.4 Working with the file system

Standard C doesn't offer much for file systems. You can only delete and rename files.

```
class EX_DIR5
```

```
inherit

    STDC_FILE_SYSTEM

creation

    make

feature

    make is
        do
            rename_to ("qqtest.abc.tmp", "qqtest.xyz.tmp")
            remove_file ("qqtest.xyz.tmp")
        end

end
```

The BON diagram is shown in **figure 7.2**.

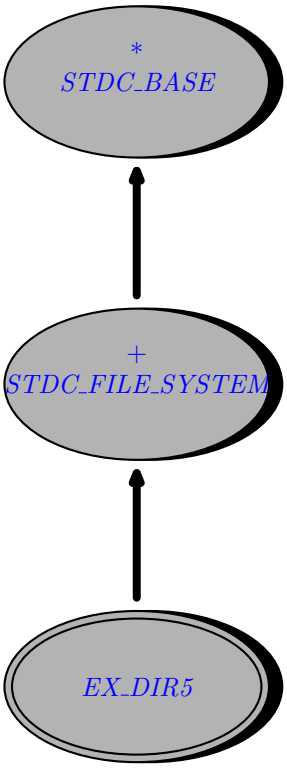

Figure 7.2 BON diagram of deleting and renaming files with Standard C.

7.5 Catching a signal

You can catch signals with Standard C. The following example demonstrates a program that can be safely interrupted by pressing Ctrl+C:

```
class EX_SIGNAL3

inherit

 EPX_CURRENT_PROCESS

 STDC_CONSTANTS

 STDC_SIGNAL_HANDLER

creation

 make

feature

 handled: BOOLEAN

 make is
  local
   signal: STDC_SIGNAL
  do
   create signal.make (SIGINT)
   signal.set_handler (Current)
   signal.apply

   print ("Wait 10s or press Ctrl+C.%N")
   sleep (10)
   if handled then
    print ("Ctrl+C pressed.%N")
   else
    print ("Ctrl+C not pressed.%N")
   end
  end

 signalled (signal_value: INTEGER) is
  do
   handled := True
  end

end
```

As Standard C doesn't have a sleep command, this program uses *EPX_CURRENT_PROCESS* to get either the *sleep* from POSIX or from Windows.

More explanation about the program itself can be found in **section 5.1**.

In this chapter:

8 Writing CGI programs

Although writing a CGI program doesn't really belong to POSIX, they still are written often, so I decided to include a few classes to make this easier. And of course, they build upon the Standard C classes.

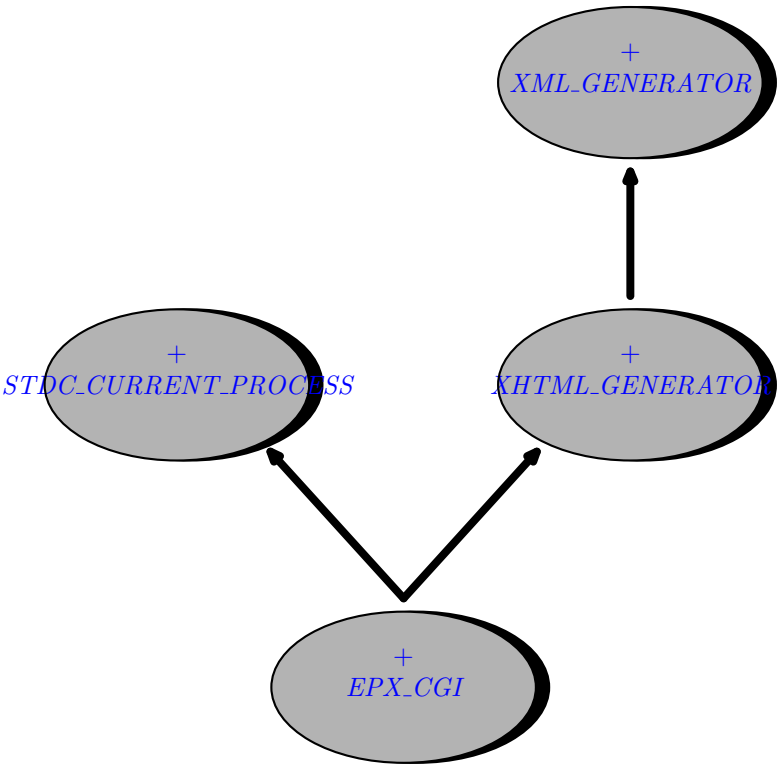

Figure 8.1 BON diagram of *EPX_CGI*.

You just inherit from *EPX_CGI* and start calling its features.

```
class EX_CGI1

inherit

    EPX_CGI

creation
```

```
    make

feature

    execute is
        do
            content_text_html

            doctype
            b_html

            b_head
            title ("e-POSIX CGI example.")
            e_head

            b_body

            p ("Hello World.")
            extend ("<p>you can use your <b>own</b> tags.</p>")
            b_p
            puts ("or use any tag by using:")
            e_p

            start_tag ("table")
            set_attribute ("border", Void)
            set_attribute ("cols", "3")
            start_tag ("tr")
            start_tag ("td")
            add_data ("start_tag")
            stop_tag
            start_tag ("td")
            add_data ("stop_tag")
            stop_tag
            stop_tag
            stop_tag

            e_body
            e_html

        end

end
```

Output is written to stdout, but also made available in *my_xml*. Generated output is XHTML, which usually displays fine with older browsers. If strict XHTML is problematic, you can call *doctype_transitional* instead of *doctype*.

It is important not to mix writing to stdout with the features you inherit from *EPX_CGI*. *EPX_CGI* does some caching, so after a tag is started by *EPX_CGI.start_tag* it is not yet written to standard

output. If you want to write something to standard output, use the *EPX_CGI.add_data* feature or its shortcut alias *puts*. If you want to write real tags, use *extend*. This last feature allows you to write anything, while *puts* escapes reserved characters like '>'.

If you use provided features like *b_a*, *b_p* and such, an attempt is made to produce good looking source. Also your input is somewhat validated against XHTML standards.

It is also easy to write a CGI program that displays a form and accepts submitted values. Even file upload is supported. The following example uses the GET method to submit data:

```
class EX_CGI2

inherit

 EPX_CGI

creation

 make

feature

 execute is
  do
   content_text_html

   doctype
   b_html

   b_head
   title ("e-POSIX CGI form example.")
   e_head

   b_body

   b_form_get ("ex_cgi2.bin")

   b_p
   puts ("Name: ")
   b_input ("text", "name")
   set_attribute ("size", "32")
   e_input
   e_p

   b_p
   puts ("City: ")
   input_text ("city", 40, "enter city here")
   e_p
```

```
    b_p
    b_button_submit ("action", "GO!")
    e_button_submit

    nbsp

    button_reset
    e_p

    e_form

    hr

    p ("In your last submit you entered:")
    b_p
    if not has_key ("name") then
     puts ("!!!!!")
    end
    puts ("name: ")
    puts (value ("name"))
    puts (", ")
    puts ("city: ")
    puts (raw_value ("city"))
    e_p

    e_body
    e_html

  end

end
```

You can use *EPX_CGI.b_input* to start an input element as shown for the input of a name. Or you can use *input_text* to start a simple text input as shown for the input of a city. Below the line you see the value a user has submitted, if any. Use *value* to get values with certain meta–characters removed. The output is still not save to be passed straight to a Unix Shell though! You can use *raw_value* to get the contents as submitted by the user.

In the above example it doesn't matter much if you use *b_form_get* or *b_form_post*. But with the GET method, you cannot upload files. The following example demonstrates how files can be uploaded:

```
class EX_CGI3

inherit

 EPX_CGI

creation
```

```
  make

feature

  execute is
    do
      content_text_html

      assert_key_value_pairs_created
      save_uploaded_files

      doctype
      b_html

      b_head
      title ("e-POSIX CGI file upload example.")
      e_head

      b_body

      b_form ("post", "ex_cgi3.bin")
      set_attribute ("enctype", multipart_encoding)

      b_p
      puts ("Filename: ")
      b_input ("file", "filename")
      set_attribute ("size", "32")
      set_attribute ("maxlength", "128")
      e_input
      e_p

      b_p
      b_button_submit ("action", "Upload file(s)")
      e_button_submit

      nbsp

      button_reset
      e_p

      e_form

      e_body
      e_html

    end
```

```
  save_uploaded_files is
   local
    kv: EPX_KEY_VALUE
    buffer: STDC_BUFFER
    target_name: STRING
    target: STDC_BINARY_FILE
   do
    create buffer.allocate (8192)
    from
     cgi_data.start
    until
     cgi_data.after
    loop
     kv := cgi_data.item_for_iteration
     if kv.file /= Void then
      from
       target_name := "/tmp/" + kv.value
       create target.create_write (target_name)
       kv.file.read_buffer (buffer, 0, 8192)
      until
       kv.file.eof
      loop
       target.write_buffer (buffer, 0, kv.file.last_read)
       kv.file.read_buffer (buffer, 0, 8192)
      end
      target.close
      kv.file.close
     end
     cgi_data.forth
    end
    buffer.deallocate
   end

end
```

It is important to set the encoding type. This example accepts a file and writes it to `/tmp`. Because multiple files can be present, this example just loops over all key value pairs and checks if a file is present. This example isn't fool–proof with multiple users submitting the same file, but you should get the idea.

Note that the first line is *EPX_CGI.content_text*: in case an exception occurs, the web server is still able to output something back to the user.

After that we make sure that the key value pairs are created with *assert_key_value_pairs_created*. They are automatically created if you call *value*, but in this case we want the key value pairs themselves. In *EX_CGI3.save_uploaded_files* we use the *EPX_KEYVALUE.file* feature to check if that key value pair is an uploaded file: if it is not Void, it points to a temporary file. As this file will be deleted when it is closed or when your program exits, we have to copy it to a new file.

The filename is just the value part of this key value pair. The filename is guaranteed to be free of directory parts.

In the last example we just print all key/value pairs to the file `list.txt` in the temporary directory. We redirect the user to another file.

```
class EX_CGI4

inherit

 EPX_CGI

 EPX_FACTORY

creation

 make

feature

 execute is
  do
   assert_key_value_pairs_created
   save_values

   extend ("Location: /mydir/myfile.html")
   new_line
   new_line
  end

 save_values is
  local
   fout: STDC_TEXT_FILE
   kv: EPX_KEY_VALUE
  do
   create fout.create_write (fs.temporary_directory + "/list.txt")
   from
    cgi_data.start
   until
    cgi_data.after
   loop
    kv := cgi_data.item_for_iteration
    fout.puts (kv.key)
    fout.puts ("%T")
    fout.puts (kv.value)
    fout.puts ("%N")
    cgi_data.forth
   end
```

```
    fout.close
  end

end
```

In this chapter:

- ***Compiling POSIX programs in Windows***
- ***Native Windows***
- ***Binary mode versus text mode***

9 e-POSIX in Windows

e-POSIX offers three alternatives to writing programs that run on both Unix and Windows platforms:

1. Write programs that only rely on Standard C. If you use only Standard C classes your program is probably quite portable. Standard C doesn't offer that much however.

2. Write programs that are based on POSIX. You use a POSIX emulator to compile and run your program unchanged on Windows. The only thing you have to be aware of is the distinction between binary and text files.

3. Write programs that are based upon e-POSIX's EPX_XXXX layer. This layer is based on e-POSIX's ABSTRACT_XXXX classes, that covers code that is common between Windows and a POSIX platform.

   Previous versions of e-POSIX used a factory class approach to access this common code. This is no longer needed. The ABSTRACT_XXXX are maded effective through EPX_XXXX classes when compiling for Windows or for POSIX.

The following sections offer more details about the last two approaches.

9.1 Compiling POSIX programs in Windows

You can also use a very large subset of POSIX under Windows with a POSIX emulator. I've tested this using SmallEiffel and Cygwin's freely available emulator. Here the steps:

1. Download the Cygwin toolkit from `http://sources.redhat.com/cygwin`.

2. Set the compiler in `compiler.se` to `gcc`. Leave the system in `system.se` to Windows.

3. Configure e-POSIX as described in **1.2** and create `libeposix_se.a`

A few things are not available under Cygnus' POSIX emulation:

1. *POSIX_FILE_SYSTEM.create_fifo* is not supported. Any attempt to use it will return `ENOSYS`. I'm not sure if returning an error is the correct solution for applications that require POSIX compatibility, because you are only warned at run–time. Another solution would be to include a call to `mkfifo`and if you use it, let the linker complain.

2. There is no locking, so calls to *POSIX_FILE_DESCRIPTOR.get_lock* and such will fail.

3. Certain POSIX tests assume that a more Unix like environment is available, so not all tests will run. For example the standard Cygwin distribution doesn't have a `more` utility. If you make a symbolic link from `less` to `more` the child process test will run.

4. The current list of implemented functions is available from `http://sources.redhat.com/cygwin/faq/faq_3.html#SEC17`.

9.2 Native Windows

Previous versions of e-POSIX used a factory class approach to access Windows or POSIX specific code. This is obsolete.

If you want to write code that is portable between Windows and POSIX use the EPX_XXXX class layer. For example you can use the *EPX_FILE_DESCRIPTOR* to use file descriptors that are completely portable between these two OSes. Use *EPX_FILE_SYSTEM* to have access to file system specific code to change directories or get the temporary directory.

In general you can replace the POSIX_ prefix with EPX_ to compile most of the examples presented in the previous POSIX specific chapters. The classes currently available in the EPX_XXXX layer are:

- *EPX_CURRENT_PROCESS*.
- *EPX_EXEC_PROCESS*.
- *EPX_FILE_DESCRIPTOR*.
- *EPX_FILE_SYSTEM*.
- *EPX_PIPE*.

Figure one shows hoe the *EPX_FILE_DESCRIPTOR* class is derived from *ABSTRACT_FILE_DESCRIPTOR*. Both Windows and POSIX have an effective *EPX_FILE_DESCRIPTOR* class. Classes as *POSIX_FILE_DESCRIPTOR* implement POSIX specific functionality for a file descriptor.

An example of using the *EPX_FILE_SYSTEM* class is shown below:

class *EX_EPX1*

inherit

EPX_FILE_SYSTEM

creation

make

feature

make **is**
local

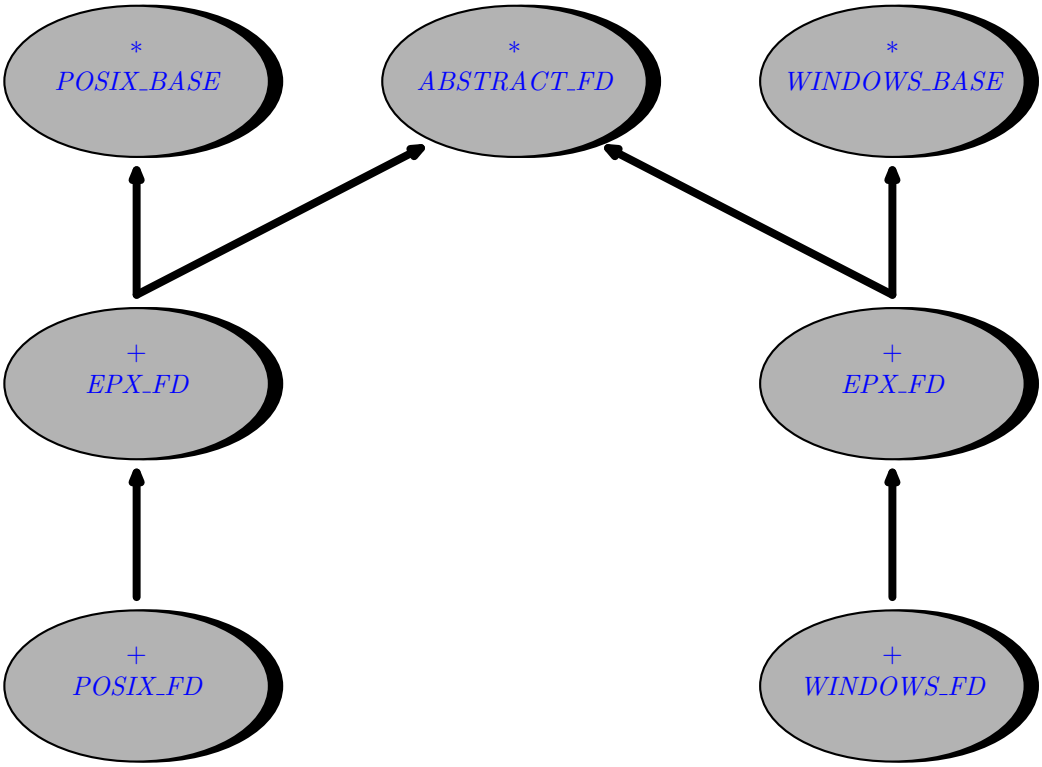

Figure 9.1 How EPX_XXXX classes are related to the POSIX and Windows classes

```
        dir: STRING
    do
        print ("Current directory: ")
        dir := current_directory
        print (dir)
        print ("%N")
        change_directory ("..")
        change_directory (dir)
        make_directory ("abc")
        rename_to ("abc", "def")
        remove_directory ("def")
    end

end
```

In **appendix C** all abstract classes are listed. There deferred features are made effective in the EPX class for the operating system you're compiling for.

9.3 Binary mode versus text mode

Independent of what layer you use to write Windows programs, you have to deal with binary and text modes. And if you usually write Unix programs and want them to work on Windows too, you have to bother with it too.

On Windows, each line of a text files ends with a carriage return character followed by a line feed character. In Eiffel this is the string "%R%N". If you use streams, *STDC_TEXT_FILE* and *STDC_BINARY_FILE*, this distinction is transparently handled.

For file descriptions, it depends. If you use Cygwin, file descriptors can either be binary or text. This depends on Cygwin settings. If you use the EPX layer, file descriptors are binary, period. The reason for this is that the underlying code is the Win32 file API which is binary only. This is usually no problem. Reading a line will still stop when a '%N' is encountered. If you use the *ABSTRACT_FILE_DESCRIPTOR.chop_last_string* method, it automatically removes any '%R' if there was such a character just before the '%N'.

At this moment, there is no explicit support for creating text files using file descriptors with the proper Windows end of file characters. *STDC_TEXT_FILE* works fine here work fine here.

If you want to use binary standard input or binary standard output, use the file descriptors available in *EPX_CURRENT_PROCESS* as *fd_stdin* and *fd_stdout*.

For Cygwin users, the following information can be helpful to get the binary versus text file distinction correct:

- Mount the volume in binary mode.
- Set the environment variable CYGWIN to 'binary'.

More information about Cygwin and CR/LF handling can be found at http://sources.redhat.com/cygwin/faq/faq_toc.html#TOC62.

In this chapter:

- ***Error handling with exceptions***
- ***Manual error handling***

10 Error handling

This chapter describes certain error handling strategies that are possible with e-POSIX. Basically there are two strategies: using the Eiffel exception mechanism or doing the error handling all yourself.

10.1 Error handling with exceptions

The opinion of the author of e-POSIX is that Eiffel's exception mechanism is very well suited to deal with things like files that cannot be opened or directories that do not exist. Others disagree, see **section 10.2**. e-POSIX is designed such that when a POSIX routine returns an error code, an exception is thrown. Here my arguments why I favor this style of error handling:

1. We all know that exceptions are to be used for breach of contract. This idea is formulated in [43]and is the best expressed opinion of exception handling I know.

   So if you ask an e-POSIX method to open a file, it will do that for you. If it cannot open the file, for whatever reason, it will raise an exception. The same argument hold if you ask it to go to a directory, to start a program, or to open a connection to another machine.

   This approach is also reflected in the names of e-POSIX's features. The name is *POSIX_TEXT_FILE.open_read* and not *POSIX_TEXT_FILE.attempt_open_read*.

2. It is usually not wise to trust clients with error handling. The larger a distance between a software failure and the error report, the more difficult it is to make a correct diagnosis of what went wrong (see [43]). e-POSIX uses the fail early, fail hard approach.

3. Error handling is often forgotten or left to some global general error handling mechanism. In an interesting article (see [43]) James Whittaker describes how he modified certain system calls to return legitimate, but unexpected return codes. Memory allocation failed for example, or opening a file returned with no more file handles. Applications failed within seconds, but it was usually completely unclear why.

4. It's a lot easier for programmer's. You don't have to write any error handling. If your program completed, you know that there wasn't a single system call that failed, that you didn't continue despite some error. This will make it possible to write programs that do their work correctly if no errors occur, or else do nothing.

First an example. Let's take a look at the code you have to write in case you want to handle failure of opening a file:

```
class EX_ERROR1

inherit
```

```
    POSIX_CURRENT_PROCESS

creation

    make

feature

    make is
        local
            fd: POSIX_FILE_DESCRIPTOR
        do
            fd := attempt_create_file
        end

    attempt_create_file: POSIX_FILE_DESCRIPTOR is
        local
            attempt: INTEGER
            still_exists: BOOLEAN
        do
            create Result.create_with_mode ("myfile", O_CREAT+O_TRUNC+O_EXCL, 0)
        rescue
            still_exists := errno.value = EEXIST
            attempt := attempt + 1
            if still_exists and then attempt <= 3 then
                sleep (1)
                retry
            end
        end

end
```

In this example we try to create a file exclusively. The create will fail if the file already exists. In case this happens, we retry 3 times. Before retrying we wait 1 second. Note that if the error is not `EEXIST`, we fail directly, without retrying.

In my opinion above's code is just the code you want to write usually: do not worry about errors, if something goes wrong, your application will fail.

My preferred way of error handling is (or sometimes should be) also reflected in the preconditions. For example the *POSIX_FILE_SYSTEM.browse_directory* has the precondition that the given path should exist and should be a directory. Quite reasonable I think. The argument against such preconditions is that it is somewhat strange: if a client has honoured the precondition by checking that the directory exists, it should be able to assume that it safely can call the routine. But between its own check and the actual call, the directory can be removed by another process.

This is the concurrent precondition paradox (see [43]). In my opinion it would not be wise to remove this precondition. It is true that honouring it, will not make sure the contract is not broken. But it still serves a very usefull purpose: documentation.

For example the routine *POSIX_FILE_SYSTEM.remove_file* does not have the precondition that the file should exist. That isn't an oversight. This routine does not fail if the file no longer exists for good reason: it honours its postcondition after all. So when you call this routine, the file may or may not exist. The routine doesn't care.

10.2 Manual error handling

In spite of the arguments listed in the previous section, automatic error handling is perhaps tedious to use when you expect a lot of errors. And some programmers just do not like Eiffel's exception mechanism. Therefore e-POSIX implements a completely different style of error handling. In this case, e-POSIX continues when an error occurs, but it safes the errorcode, and you can check the errorcode of the first error when you wish. This first errorcode has to be reset by the programmer. An example:

```
class EX_ERROR2

inherit

    STDC_SECURITY_ACCESSOR

creation

    make

feature

    make is
        local
            fd: POSIX_FILE_DESCRIPTOR
        do
            security.error_handling.disable_exceptions
            create fd.create_write ("myfile")
            if fd.errno.first_value = 0 then
                fd.write_string ("1%N")
                fd.write_string ("2%N")
                fd.close
            else
                fd.errno.clear_first
            end
        end

end
```

Exception handling is turned off by a call to *STDC_SECURITY_ACCESSOR.security.error_handling.disable_exceptions*. It can be enabled again by calling *security.error_handling.enable_exceptions*. In between, you're on your own, just like a C programmer. If `myfile` cannot be opened, nothing happens, and the *POSIX_FILE_DESCRIPTOR.write_string* feature is called. Depending if you have enabled precondition checking or not, *write_string* will fail. The precondition if *write_string*

is that the file has to be open. Therefore, at certain points, you're still forced to deal with errors. Every object has an *errno* variable. This variable points to the global *STDC_ERRNO* object (its a once routine). So there basically is just one *first_value* error value. Whatever object caused the error, you can check the *errno.first_value* of any e-POSIX object. The last error is still available in *errno.value*.

If there is no error, the program continues writing. If *POSIX_FILE_DESCRIPTOR.write_string* failed, the next one is still executed. If there is an error, we reset it with *STDC_ERRNO.clear_first*. This gives us the chance to catch another error value if an error occurs. If this method is not called, *first_value* will keep its original value.

The following example is the same as *EX_ERROR1*. It shows how to open a file exclusively with manual error handling.

```
class EX_ERROR3

inherit

    POSIX_CURRENT_PROCESS

    EXCEPTIONS

creation

    make

feature

    make is
        local
            fd: POSIX_FILE_DESCRIPTOR
        do
            security.error_handling.disable_exceptions
            fd := attempt_create_file
        end

    attempt_create_file: POSIX_FILE_DESCRIPTOR is
        require
            manual_error: not security.error_handling.exceptions_enabled
        local
            attempt: INTEGER
            still_exists: BOOLEAN
        do
            from
                attempt := 1
                still_exists := True
            until
                not still_exists or else attempt > 3
            loop
```

```
            create Result.create_with_mode ("myfile", O_CREAT+O_TRUNC+O_EXCL, 0)
            still_exists := errno.first_value = EEXIST
            if still_exists then
                sleep (1)
                attempt := attempt + 1
            end
        end
        if still_exists then
            raise ("failed to create file")
        end
    end

end
```

As you can see, manual error handling does not necessarily translate into less code.

The summary of this section is that you chould check each distinctive step when using manual error handling. You don't have to check intermediate steps.

In this chapter:

- ***Denial of service attacks***
- ***Authorization bypass attacks***

11 Security

e-POSIX is well–suited to write server applications like CGI scripts and daemons. As these applications can be hosted on servers that are attached to the Internet, they could be prone to attack. Applications written with e-POSIX could be misused in a denial of service attack or to gain root access. e-POSIX offers certain protection mechanisms that enable your applications to fend off such penetrations.

This chapter shows you how applications can be misused and what mechanisms e-POSIX offers for certain attacks.

> "Programmers typically focus on "positive" aspects of programs, that is, what is the functionality required for the task to be accomplished. Programmers rarely focus on the negative aspects of programs, that is, what functionality is not required for the program to accomplish its task. Attackers take advantage of proggrammers failure to consider negative functionality. Perhaps a reason that programmers avoid negative functionality is that there is no good way to specify what a program should not be permitted to do."

11.1 Denial of service attacks

In a denial of service attack, crackers attempt to deplete one or more finite resources. Resources can be software related like database connections or TCP/IP connections, but ultimately resources are finite because of hardware limitations. This manual distinguishes the following hardware resources:

- Memory.
- CPU.
- Disk space.
- Network bandwidth.

A denial of service attack succeeds if a cracker depletes these resources in such a way that the server cannot handle request anymore, or handles them very slowly. For example, Linux 2.2 is easy to bring to its knees if you keep on allocating memory. In normal situations your application runs fine, and allocates only a limited amount of memory. But an attacker might have found a way to make your application allocate much more memory. Even if you are sure that the code you have written is not prone to such an attack, you might use a library based on e-POSIX that does have code that is exploitable.

e-POSIX has some limited support to set limits on memory, file handle (a memory issue) and cpu usage. When a set limit has been exceeded, an exception is raised.

To limit the amount of memory, inherit from *STDC_SECURITY_ACCESSOR* and call *security.memory.set_max_allocation*. Currently this limits the amount of memory that can be allocated with

STDC_BUFFER. It does not limit the amount of memory that is allocated by *STRING* or other classes. You can also limit the amount of memory that can be allocated with a single call by calling *security.memory.set_max_single_allocation*.

You can limit the number of file handles a program can open by calling *security.files.set_max_open_files*. This works only with files and sockets opened by e-POSIX classes as *STDC_FILE* and *POSIX_FILE_DESCRIPTOR*, not with files opened through other means. In this case you cannot rely on the garbage collection to close your file. Certain garbage collectors do not allow calling other classes in the *MEMORY.dispose* method. e-POSIX needs to do this to decrement its idea of the number of open handles. Only when you explicitly call *STDC_FILE.close* will the e-POSIX decrease its open file handles.

You can limit the amount of CPU time by calling *security.cpu.set_max_process_time*. It is not possible to automatically halt your application when this time has exceeded. You have to call *security.cpu.check_process_time* to actually check the processor time used.

Currently e-POSIX cannot check disk space or network bandwidth limitations.

`Discuss here that decrementing only works for manual deallocations, I'm very sorry about that, but this is a problem of ISE. I'm thinking about ways to work around this.`

11.2 Authorization bypass attacks

A hacker can bypass authorization if he or she, through your program, can gain the following access:

- Access to more information than your program is written to provide. Security is not breached here, but your program is used in an 'innovative' way. Note that if your program runs within the root security context (suid root), security can be breached!
- Security is breached when your program is used to get more access rights than your program is written to provide. Especially suid root programs are an attractive target here.

Usually Eiffel programs do not allocate buffers on the stack, so they are not prone to the so called 'buffer overflow' attack. As certain vendors might provide some 'native' class that allocate things on the stack, leave precondition checking always on in suid root programs.

Currently e-POSIX doesn't offer much protection for suid root programs. Much better security will be the topic of a next release.

In this chapter:

- ***Making C Headers available to Eiffel***
- ***Distinction between Standard C and POSIX headers***
- ***C translation details***

12 Accessing C headers

This chapter explains the conventions that e-POSIX uses to access the C–headers.

12.1 Making C Headers available to Eiffel

The most portable and safest header translation comes when a C function is not called verbatim, but instead a translation function is used. For example to make the Standard C function `fopen`available within Eiffel a new header file is created which lists an Eiffel compatible way to call this routine:

```
#include "eiffel.h"
#include <stdio.h>

EIF_POINTER posix_fopen(EIF_POINTER filename, EIF_POINTER mode);
```

Instead of using C types, we use Eiffel types here, which are made available by including `eiffel.h`.

The corresponding C file contains the following implementation:

```
#include "my_new_header.h"

EIF_POINTER posix_fopen(EIF_POINTER filename, EIF_POINTER mode)
{
  return ( (EIF_POINTER) fopen (filename, mode));
}
```

It simply calls the original function, returning the result. Type conversion between Eiffel and C types shouldn't pose problems this way.

To be able to call this function from Eiffel, an **external** feature needs to be written. For example:

```
class HEADER_STDIO

feature {NONE} -- C binding for stream functions

   posix_fopen (path, a_mode: POINTER): POINTER is
         -- Opens a stream
      require
         valid_mode: a_mode /= default_pointer
      external "C"
      end

end
```

Of course, the Eiffel function can have all Design By Contract features Eiffel programmers are accustomed too.

To recapitulate: every header that is to be translated, needs:

1. a new header file, and
2. a corresponding C file, and
3. an Eiffel class.

For example to translate <stdio.h> a header file like `eiffel_stdio.h` and a C file `eiffel_stdio.c` is needed. The Eiffel class could be in `header_stdio.e`.

12.2 Distinction between Standard C and POSIX headers

However, POSIX sometimes defines extensions to existing Standard C headers. Simply using a translation header file like `eiffel_stdio.h` will not work for pure Standard C Eiffel programs, as it can include POSIX specific extensions that might simply not be available on a given platform.

Therefore, e-POSIX divides the C headers in several groups:

1. The Standard C headers.
2. The POSIX headers.
3. The Single Unix Specification headers.
4. Microsoft Windows headers (as far as they define POSIX functions, this library does not translate Microsoft Windows specific functions).

Every group gets its own translation header with its own prefix. A translated header has a prefix, an underscore and next the original header name. The Standard C translation of <stdio.h> is done in `c_stdio.h` and `c_stdio.c`. The POSIX extensions to this header are available in `p_stdio.h` and `p_stdio.c`.

The corresponding Eiffel class follows similar conventions. It has the group's prefix, next the string 'API', an underscore and next the name of the header. So all <stdio.h> functions are made available in *CAPI_STDIO*.

In **table 12.1** all the groups with there translation header prefix and Eiffel class prefix are listed. See also the directory structure in **figure 12.1**.

12.3 C translation details

This translation wants to do as less as possible at the C level. It attempts to just make available the C constants and C functions and do the actual work in Eiffel.

A few details:

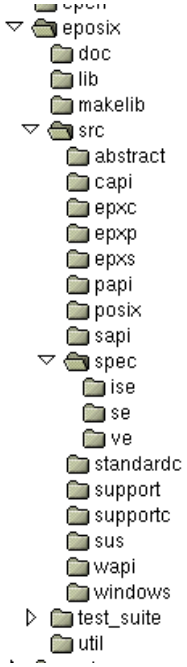

Figure 12.1 e-POSIX directory structure

Group	directory	header prefix	class prefix
Standard C	`src/capi`	c	CAPI
POSIX	`src/[api`	p	PAPI
Single Unix Specification	`src/sapi`	s	SAPI
Windows	`src/wapi`	w	WAPI

Table 12.1 e-POSIX prefix conventions

1. Constants, C macro definitions, are exported in the header file with the prefix ‘const_’ and next the macro name. The Eiffel API class exports these constants with the original, uppercased name.

2. Struct members are exported with getter and setter functions. The get function has the prefix ‘posix’, an underscore, the struct name, an underscore and as last the member name. The set function has the prefix ‘posix’, an underscore, ‘set’, an underscore, the struct name, an underscore and as last the member name.

In this chapter:

A Posix function to Eiffel class mapping list

The following table defines exactly where a given Posix function is used in a Eiffel class mapping. The table is sorted in alphabetic order. Note that when a STDC_ class is listed, the feature is also available in the corresponding POSIX_ class.

Function	Header	Class	Comment
abort	<stdlib.h>	*STDC_CURRENT_PROCESS.abort*	
access	<unistd.h>	*ABSTRACT_FILE_SYSTEM.is_accessible*	
aio_cancel	<aio.h>	*POSIX_ASYNC_IO_REQUEST.cancel*	
aio_error	<aio.h>	*POSIX_ASYNC_IO_REQUEST.is_pending*	
aio_fsync	<aio.h>	*POSIX_ASYNC_IO_REQUEST.synchronize*	
aio_read	<aio.h>	*POSIX_ASYNC_IO_REQUEST.read*	
aio_return	<aio.h>	*POSIX_ASYNC_IO_REQUEST.return_status*	
aio_suspend	<aio.h>	*POSIX_ASYNC_IO_REQUEST.wait_for*	
aio_write	<aio.h>	*POSIX_ASYNC_IO_REQUEST.write*	
alarm	<unistd.h>	*POSIX_TIMED_COMMAND*	
asctime	<time.h>	*STDC_TIME.default_format*	
atexit	<stdlib.h>		probably not applicable.
calloc	<stdlib.h>	*STDC_BUFFER.allocate_and_clear*	
cfgetispeed	<termios.h>	*POSIX_TERMIOS.input_speed*	
cfgetospeed	<termios.h>	*POSIX_TERMIOS.output_speed*	
cfsetispeed	<termios.h>	*POSIX_TERMIOS.set_input_speed*	
cfsetospeed	<termios.h>	*POSIX_TERMIOS.set_output_speed*	
chdir	<unistd.h>	*POSIX_FILE_SYSTEM.change_directory*	
chmod	<sys/stat.h>	*POSIX_FILE_SYSTEM.change_mode*	
chown	<unistd.h>	*POSIX_PERMISSIONS_PATH.apply_owner_and_group*	
clearerr	<stdio.h>	*STDC_FILE.clear_error*	
clock	<time.h>	*STDC_CURRENT_PROCESS.clock*	
clock_getres	<time.h>		
clock_gettime	<time.h>		
clock_settime	<time.h>		
close	<unistd.h>	*POSIX_FILE_DESCRIPTOR.close*	
closedir	<dirent.h>	*POSIX_DIRECTORY*	
creat	<fcntl.h>	*POSIX_FILE_DESCRIPTOR.create_read_write*	
ctermid	<unistd.h>		
ctime	<time.h>		Can be emulated with *STDC_TIME*.
cuserid	<stdio.h>		see `getlogin`
difftime	<time.h>	*STDC_TIME*	
dup	<unistd.h>	*POSIX_FILE_DESCRIPTOR.make_as_duplicate*	
dup2	<unistd.h>	*POSIX_FILE_DESCRIPTOR.make_as_duplicate*	
execl	<unistd.h>		See `execvp`.
execle	<unistd.h>		See `execvp`.
execlp	<unistd.h>		See `execvp`.
execv	<unistd.h>		See `execvp`.
execve	<unistd.h>		See `execvp`.

execvp	<unistd.h>	*POSIX_EXEC_PROCESS.execute*	
exit	<stdlib.h>	*STDC_CURRENT_PROCESS.exit*	
_exit	<unistd.h>		
fclose	<stdio.h>	*STDC_FILE.close*	
fcntl	<unistd.h>	*POSIX_FILE_DESCRIPTOR*	*attempt_lock*,*get_lock*, *set_lock* and others.
fdatasync	<unistd.h>	*POSIX_FILE_DESCRIPTOR.synchronize_data*	This function is not available on many so called POSIX systems. In such cases it is mapped to `fsync`.
fdopen	<stdio.h>	*POSIX_FILE.make_from_file_descriptor*	
feof	<stdio.h>	*STDC_FILE.eof*	
ferror	<stdio.h>	*STDC_FILE.error*	
fflush	<stdio.h>	*STDC_FILE.flush*	
fgetc	<stdio.h>	*STDC_FILE.get_character*	
fgetpos	<stdio.h>	*STDC_FILE.get_position*	
fgets	<stdio.h>	*STDC_FILE.get_string*	
fileno	<stdio.h>	*POSIX_FILE_DESCRIPTOR.make_from_file*	
fopen	<stdio.h>	*STDC_FILE*	various open creation features.
fork	<unistd.h>	*POSIX_CURRENT_PROCESS.fork*	
fpathconf	<unistd.h>		
fprintf	<stdio.h>		not applicable.
fputc	<stdio.h>	*STDC_FILE.putc*	
fputs	<stdio.h>	*STDC_FILE.put_string*	
fread	<stdio.h>	*STDC_FILE.read*	Also *read_string* and *read_character*.
free	<stdlib.h>	*STDC_BUFFER.deallocate*	
freopen	<stdio.h>	*STDC_FILE.reopen*	
fseek	<stdio.h>	*STDC_FILE.seek*	Also *seek_from_current* and *seek_from_end*.
fsetpos	<stdio.h>	*STDC_FILE.set_position*	
fstat	<sys/stat.h>	*POSIX_STATUS*	Returned by *POSIX_FILE_DESCRIPTOR.status*.
fsync	<unistd.h>	*POSIX_FILE_DESCRIPTOR.synchronize*	
ftell	<stdio.h>	*STDC_FILE.tell*	
fwrite	<stdio.h>	*STDC_FILE.write*	
getc	<stdioh>		See `fgetc`.
getchar	<stdio.h>		See `fgetc`.
getcwd	<unistd.h>	*POSIX_FILE_SYSTEM.current_directory*	
getegid	<unistd.h>	*POSIX_CURRENT_PROCESS.effective_group_id*	
getenv	<stdlib.h>	*STDC_ENV_VAR.value*	
geteuid	<unistd.h>	*POSIX_CURRENT_PROCESS.effective_user_id*	
getgid	<unistd.h>	*POSIX_CURRENT_PROCESS.real_group_id*	
getgrgid	<grp.h>	*POSIX_GROUP.make_from_gid*	
getgrnam	<grp.h>	*POSIX_GROUP.make_from_name*	
getgroups	<unistd.h>	*POSIX_CURRENT_PROCESS.is_in_group*	
getlogin	<unistd.h>	*POSIX_CURRENT_PROCESS.login_name*	
getpgrp	<unistd.h>	*POSIX_CURRENT_PROCESS.process_group_id*	
getpid	<unistd.h>	*POSIX_CURRENT_PROCESS.pid*	
getppid	<unistd.h>	*POSIX_CURRENT_PROCESS.parent_pid*	
getpwnam	<pwd.h>	*POSIX_USER.make_from_name*	
getpwuid	<pwd.h>	*POSIX_USER.make_from_uid*	
gets	<stdio.h>		See `fgets`.
gettimeofday	<sys/time.h>		*SUS_TIME_VALUE*
getuid	<unistd.h>	*POSIX_CURRENT_PROCESS.real_user_id*	
gmtime	<time.h>	*STDC_TIME.to_utc*	
isatty	<unistd.h>	*POSIX_FILE_DESCRIPTOR.is_attached_to_terminal*	
kill	<signal.h>	*POSIX_PROCESS.kill*	
link	<unistd.h>	*POSIX_FILE_SYSTEM.link*	

lio_listio	<aio.h>		
localeconv	<locale.h>	*STDC_LOCALE_NUMERIC*	
localtime	<time.h>	*STDC_TIME.to_local*	
lseek	<unistd.h>	*POSIX_FILE_DESCRIPTOR.seek*	Also *seek_from_current* and *seek_from_end*.
malloc	<stdlib.h>	*STDC_BUFFER.allocate*	
memcpy	<string.h>	*STDC_BUFFER.memory_copy*	See also *copy_from*.
memchr	<string.h>		
memcmp	<string.h>		
memmove	<string.h>	*STDC_BUFFER.memory_move*	
memset	<string.h>	*STDC_BUFFER.fill_with*	
mkdir	<sys/stat.h>	*POSIX_FILE_SYSTEM.make_directory*	
mkfifo	<sys/stat.h>	*POSIX_FILE_SYSTEM.create_fifo*	
mktime	<time.h>	*STDC_TIME.set_date_time*	Also *set_date* and *set_time*.
mlockall	<sys/mman.h>		
mlock	<sys/mman.h>		
mmap	<sys/mman.h>	*POSIX_MEMORY_MAP*	
mprotect	<sys/mman.h>		
mq-receive	<mqueue.h>		
mq_close	<mqueue.h>		
mq_getattr	<mqueue.h>		
mq_notify	<mqueue.h>		
mq_open	<mqueue.h>		
mq_send	<mqueue.h>		
mq_setattr	<mqueue.h>		
mq_unlink	<mqueue.h>		
msync	<sys/mman.h>		
munlockall	<sys/mman.h>		
munlock	<sys/mman.h>		
munmap	<sys/mman.h>	*POSIX_MEMORY_MAP*	
nanosleep	<time.h>		
open	<fcntl.h>	*POSIX_FILE_DESCRIPTOR.open*	Also *open_read*, *open_read_write* and *open_write*
opendir	<dirent.h>	*POSIX_DIRECTORY*	
pathconf	<unistd.h>	*POSIX_DIRECTORY.max_filename_length*	
pause	<unistd.h>	*POSIX_CURRENT_PROCESS.pause*	
perror	<stdio.h>		e-POSIX generates exceptions on error.
pipe	<unistd.h>	*POSIX_PIPE.make*	
printf	<stdio.h>		not applicable.
putc	<stdio.h>		See `fputc`.
putchar	<stdio.h>		See `fputc`.
puts	<stdio.h>		See `fputs`.
raise	<signal.h>	*STDC_SIGNAL.raise*	
rand	<stdlib.h>	*STDC_CURRENT_PROCESS.random*	
read	<unistd.h>	*POSIX_FILE_DESCRIPTOR.read*	
readdir	<dirent.h>	*POSIX_DIRECTORY*	
realloc	<stdlib.h>	*STDC_BUFFER.resize*	
remove	<stdio.h>	*POSIX_FILE_SYSTEM.remove_file*	
rename	<unistd.h>	*POSIX_FILE_SYSTEM.rename_to*	
rewind	<stdio.h>	*STDC_FILE.rewind*	
rewinddir	<dirent.h>	*POSIX_DIRECTORY*	
rmdir	<unistd.h>	*POSIX_FILE_SYSTEM.remove_directory*	
scanf	<stdio.h>		not applicable.
sem_close	<semaphore.h>		
sem_destroy	<semaphore.h>		
sem_getvalue	<semaphore.h>		
sem_init	<semaphore.h>	*POSIX_UNNAMED_SEMAPHORE.create_shared*	And *create_unshared*.
sem_open	<semaphore.h>		

sem_post	<semaphore.h>	*POSIX_SEMAPHORE.release*	
sem_trywait	<semaphore.h>	*POSIX_SEMAPHORE.attempt_acquire*	
sem_unlink	<semaphore.h>		
sem_wait	<semaphore.h>	*POSIX_SEMAPHORE.acquire*	
setbuf	<stdio.h>	*STDC_FILE.set_buffer*	
setgid	<unistd.h>	*POSIX_CURRENT_PROCESS.set_group_id*	Also *restore_group_id*.
setlocale	<locale.h>	*STDC_CURRENT_PROCESS.set_locale*	Also *set_native_locale* and *set_native_time*.
setpgid	<unistd.h>	*PAPI_UNISTD.posix_setsid*	
setsid	<unistd.h>	*PAPI_UNISTD.posix_setsid*	
setuid	<unistd.h>	*POSIX_CURRENT_PROCESS.set_user_id*	Also *restore_user_id*.
setvbuf	<stdio.h>	*STDC_FILE.set_no_buffering*	Also *set_full_buffering* and *set_line_buffering*
shm_open	<sys/mman.h>	*POSIX_SHARED_MEMORY.open_read_write*	And *create_write*, *open_read*, Efeatureopen_write.
shm_unlink	<sys/mman.h>	*POSIX_FILE_SYSTEM.unlink_shared_memory_object*	
sigaction	<signal.h>	*POSIX_SIGNAL*	
sigaddset	<signal.h>	*POSIX_SIGNAL_SET.add*	
sigdelset	<signal.h>	*POSIX_SIGNAL_SET.prune*	
sigemptyset	<signal.h>	*POSIX_SIGNAL_SET.make_empty*	
sigfillset	<signal.h>	*POSIX_SIGNAL_SET.make_full*	
sigismember	<signal.h>	*POSIX_SIGNAL_SET.has*	
signal	<signal.h>	*STDC_SIGNAL.raise*	
sigpending	<signal.h>	*POSIX_SIGNAL_SET.make_pending*	
sigprocmask	<signal.h>	*POSIX_SIGNAL_SET.add_to_blocked_signals*	Also *remove_from_blocked_signals* and *set_blocked_signals*
sigqueue	<signal.h>		
sigsuspend	<signal.h>	*POSIX_SIGNAL_SET.suspend*	
sigtimedwait	<signal.h>		
sigwait	<signal.h>		
sigwaitinfo	<signal.h>		
sleep	<unistd.h>	*POSIX_CURRENT_PROCESS.sleep*	
sprintf	<stdio.h>		Not applicable.
srand	<stdlib.h>	*STDC_CURRENT_PROCESS.set_random_seed*	
sscanf	<stdio.h>		Not applicable.
stat	<sys/stat.h>	*POSIX_STATUS*	
strftime	<time.h>	*STDC_TIME.format*	
sysconf	<unistd.h>	*POSIX_SYSTEM*	
system	<stdlib.h>	*STDC_SHELL_COMMAND*	
tcdrain	<unistd.h>		
tcflow	<unistd.h>		
tcflush	<unistd.h>	*POSIX_TERMIOS.flush_input*	
tcgetattr	<unistd.h>	*POSIX_TERMIOS.make*	
tcgetpgrp	<unistd.h>		
tcsendbreak	<unistd.h>		
tcsetattr	<unistd.h>	*POSIX_TERMIOS.apply_now*	Also *apply_drain* and *apply_flush*
tcsetpgrp	<unistd.h>		
time	<time.h>	*STDC_TIME.make_from_unix_time*	
timer_create	<signal.h>		
timer_create	<time.h>		
times	<times.h>		
tmpfile	<stdio.h>	*STDC_TEMPORARY_FILE.make*	
tmpnam	<stdio.h>	*STDC_FILE_SYSTEM.temporary_file_name*	
ttyname	<unistd.h>	*POSIX_FILE_DESCRIPTOR.ttyname*	
tzset	<time.h>		
umask	<sys/stat.h>		
uname	<sys/utsname.h>	*POSIX_SYSTEM*	Various queries.
ungetc	<stdio.h>	*STDC_FILE.ungetc*	
unlink	<unistd.h>	*POSIX_FILE_SYSTEM.unlink*	

utime	<utime.h>	*POSIX_FILE_SYSTEM.utime*	See also its *touch* method.
vfprintf	<stdio.h>		Not applicable.
vprintf	<stdio.h>		Not applicable.
vsprint	<stdio.h>		Not applicable.
wait	<sys/wait.h>	*POSIX_CURRENT_PROCESS.wait*	
waitpid	<sys/wait.h>	*POSIX_FORK_ROOT.wait_pid*	
write	<unistd.h>	*POSIX_FILE_DESCRIPTOR.write*	

This tabel does not contain the following category of functions:

1. Math functions.
2. String functions, including wide character/multibyte string. routines. The memory move/copy functions are included, some of them even supported.
3. No type conversion functions.
4. No functions from <ctype.h>.
5. No functions from <setjmp.h>.
6. No functions from <stdarg.h>.
7. No string formatting functions like sscanf. I suggest you use the Formatter library for that. You can download this excellent library at http://www.eiffel-forum.org/archive/dominicu/format.htm.

Functions in above categories are either not applicable, already present in Eiffel or are better off in a different library.

B
Short (flat) listing of Standard C classes

In this chapter:

B.1 STDC_BASE

```
class interface STDC_BASE
feature(s) from STDC_BASE
    -- errno
    errno: STDC_ERRNO
invariant
    accessing_real_singleton: security_is_real_singleton;
end of STDC_BASE
```

B.2 STDC_BUFFER

class *interface STDC_BUFFER*
creation
allocate (*a_capacity*: *INTEGER*)
-- allocate memory of *a_capacity* bytes
allocate_and_clear (*a_capacity*: *INTEGER*)
-- allocate memory of *a_capacity* bytes, make sure its zeroed out
make_from_pointer (*a_pointer*: *POINTER*; *a_capacity*: *INTEGER*; *a_become_owner*: *BOOLEAN*)
-- attach a pointer to this object. If *a_become_owner* is
-- True, it will deallocate the pointer when *close* is
-- called, or when this object is garbage collected.
feature(*s*) **from** *STDC_BUFFER*
-- Creation
allocate (*a_capacity*: *INTEGER*)
-- allocate memory of *a_capacity* bytes
allocate_and_clear (*a_capacity*: *INTEGER*)
-- allocate memory of *a_capacity* bytes, make sure its zeroed out
make_from_pointer (*a_pointer*: *POINTER*; *a_capacity*: *INTEGER*; *a_become_owner*: *BOOLEAN*)
-- attach a pointer to this object. If *a_become_owner* is
-- True, it will deallocate the pointer when *close* is
-- called, or when this object is garbage collected.
feature(*s*) **from** *STDC_BUFFER*
-- Other allocation commands
resize (*new_capacity*: *INTEGER*)
-- Resize memory to *new_capacity* bytes. Expanded memory is not
-- guaranteed to be zeroed out.
feature(*s*) **from** *STDC_BUFFER*
-- Access
resource_usage_can_be_increased: *BOOLEAN*
-- Can the number of allocated resources increased with *capacity*?
feature(*s*) **from** *STDC_BUFFER*
-- Copy data internally or externally
copy_from (*source*: *STDC_BUFFER*; *src_offset*, *dest_offset*, *bytes*: *INTEGER*)
-- Move data from another buffer into ourselves.
-- Start at offset *src_offset*, into
-- offset *dest_offset*, moving *bytes* bytes
-- Memory may overlap.
memory_copy (*source*: *POINTER*; *src_offset*: *INTEGER*; *dest_offset*, *bytes*: *INTEGER*)
-- Copy data from *source*, offset *src_offset*, to location
-- *dest_offset* in this buffer, for *bytes* bytes.
-- Memory may not overlap, use *move* to copy within buffer
-- or *memory_move* to copy from potentially overlapping buffer.
memory_move (*source*: *POINTER*; *src_offset*: *INTEGER*; *dest_offset*, *bytes*: *INTEGER*)
-- Copy data from *source*, offset *src_offset*, to location
-- *dest_offset* in this buffer, for *bytes* bytes.
-- Memory may overlap.

move (*src_offset*, *dest_offset*: *INTEGER*; *bytes*: *INTEGER*)
-- Move data around in buffer itself from offset *src_offset* to
-- offset *dest_offset*, moving *bytes* bytes.
-- Memory may overlap.
feature(*s*) **from** *STDC_BUFFER*
-- Set/get bytes (8-bit data)
peek_uint8 (*index*: *INTEGER*): *INTEGER*
-- consider memory an array of 8 bit values.
infix "@" (*index*: *INTEGER*): *INTEGER*
-- consider memory an array of 8 bit values.
poke_uint8 (*index*, *value*: *INTEGER*)
peek_int8 (*index*: *INTEGER*): *INTEGER*
-- consider memory an array of 8 bit values.
poke_int8 (*index*, *value*: *INTEGER*)
feature(*s*) **from** *STDC_BUFFER*
-- Set/get integers (16-bit data)
peek_int16 (*index*: *INTEGER*): *INTEGER*
-- Read signed 16 bit value at offset *index* in native
-- endian format.
peek_int16_native (*index*: *INTEGER*): *INTEGER*
-- Read signed 16 bit value at offset *index* in native
-- endian format.
peek_uint16 (*index*: *INTEGER*): *INTEGER*
-- Read unsigned 16 bit value at offset *index* in native format.
peek_uint16_native (*index*: *INTEGER*): *INTEGER*
-- Read unsigned 16 bit value at offset *index* in native format.
peek_int16_big_endian (*index*: *INTEGER*): *INTEGER*
-- Read 16 bit value at offset *index* in big endian format.
peek_int16_little_endian (*index*: *INTEGER*): *INTEGER*
-- Read 16 bit value at offset *index* in little endian format.
poke_int16 (*index*: *INTEGER*; *value*: *INTEGER*)
-- Write 16 bit value at offset *index*, in native endian format.
poke_int16_native (*index*: *INTEGER*; *value*: *INTEGER*)
-- Write 16 bit value at offset *index*, in native endian format.
poke_int16_big_endian (*index*: *INTEGER*; *value*: *INTEGER*)
-- Write 16 bit value at offset *index*, in big endian format.
poke_int16_little_endian (*index*: *INTEGER*; *value*: *INTEGER*)
-- Write 16 bit value at offset *index*, in little endian format.
feature(*s*) **from** *STDC_BUFFER*
-- Set/get integers (32-bit data)
peek_int32_native (*index*: *INTEGER*): *INTEGER*
-- Read 32 bit value at offset *index*, assume its byte order
-- is native, and return it.
peek_integer (*index*: *INTEGER*): *INTEGER*
-- Read 32 bit value at offset *index*, assume its byte order
-- is native, and return it.
peek_int32_big_endian (*index*: *INTEGER*): *INTEGER*

-- Read 32 bit value at offset *index*, assume its byte order
-- is big endian, and return it in native format.
peek_int32_little_endian (*index*: *INTEGER*): *INTEGER*
-- Read 32 bit value at offset *index*, assume its byte order
-- is little endian, and return it in native format.
peek_uint32_native (*index*: *INTEGER*): *INTEGER*
-- Read 32 bit unsigned int at offset *index*, assume native
-- byte order.
peek_uint32_big_endian (*index*: *INTEGER*): *INTEGER*
-- Read 32 bit unsigned int at offset *index*, assume its
-- byte order is big endian, and return it in native format.
peek_uint32_little_endian (*index*: *INTEGER*): *INTEGER*
-- Read 32 bit unsigned int at offset *index*, assume its
-- byte order is big endian, and return it in native format.
poke_integer (*index*: *INTEGER*; *value*: *INTEGER*)
-- Write 32 bit value at offset *index*, in native endian format.
poke_int32_native (*index*: *INTEGER*; *value*: *INTEGER*)
-- Write 32 bit value at offset *index*, in native endian format.
poke_int32_big_endian (*index*: *INTEGER*; *value*: *INTEGER*)
-- Write 32 bit value at offset *index*, in big endian format.
poke_int32_little_endian (*index*: *INTEGER*; *value*: *INTEGER*)
-- Write 32 bit value at offset *index*, in little endian format.
feature(*s*) **from** *STDC_BUFFER*
-- Set/get characters
peek_character (*index*: *INTEGER*): *CHARACTER*
-- return value at position *index* as a character
poke_character (*index*: *INTEGER*; *value*: *CHARACTER*)
-- set character at *position* index to *value*
c_substring_with_string (*dest*: *STRING*; *start_index*, *end_index*: *INTEGER*)
-- As *c_substring* but used *dest* as the destination.
c_substring (*start_index*, *end_index*: *INTEGER*): *STRING*
-- Create a substring containing all characters from
-- start_index up to encountering a %U or when end_index is
-- reached, whatever happens first.
substring (*start_index*, *end_index*: *INTEGER*): *STRING*
-- Create a substring containing all characters
-- from start_index to end_index inclusive.
feature(*s*) **from** *STDC_BUFFER*
-- Fill
fill_at (*start_index*, *a_count*: *INTEGER*; *byte*: *INTEGER*)
-- Starting at position *start_index*, write *byte* for *a_count* bytes
feature(*s*) **from** *STDC_BUFFER*
-- Searching
locate_character (*other*: *CHARACTER*; *start_index*: *INTEGER*): *INTEGER*
-- Return index of *other* in buffer, or -1.
-- Search begins at *start_index*.
locate_string (*other*: *STRING*; *start_index*: *INTEGER*): *INTEGER*

```
        -- Does buffer contain other?
        -- Returns index where other is found.
        -- Returns -1 if not found
        -- searching starts at position start_index
feature(s) from STDC_BUFFER
    -- Queries
    is_valid_index (index: INTEGER): BOOLEAN
    is_valid_range (from_index, to_index: INTEGER): BOOLEAN
        -- Returns True if from_index..to_index is a valid and
        -- meaningfull range
feature(s) from STDC_BUFFER
    -- Low level handle functions
    do_close: BOOLEAN
        -- Close resource, return error if any, or zero on
        -- success. This routine may never call another object, else
        -- it cannot be used safely in dispose.
    unassigned_value: POINTER
        -- The value that indicates that handle is unassigned.
invariant
    accessing_real_singleton: security_is_real_singleton;
    capacity_not_negative: capacity >= 0;
    valid_capacity: is_allocated = capacity > 0;
    open_implies_handle_assigned: is_allocated = ptr /= unassigned_value;
    owned_implies_open: is_owner implies is_allocated;
    owned_implies_handle_assigned: is_owner implies ptr /= unassigned_value;
end of STDC_BUFFER
```

B.3 STDC_CONSTANTS

class *interface STDC_CONSTANTS*
feature(*s*) **from** *STDC_CONSTANTS*
-- error codes
edom: *INTEGER*
-- Math argument out of domain of function
erange: *INTEGER*
-- Math result not representable
feature(*s*) **from** *STDC_CONSTANTS*
-- standard streams
stream_stdin: *POINTER*
stream_stdout: *POINTER*
stream_stderr: *POINTER*
feature(*s*) **from** *STDC_CONSTANTS*
-- characters
const_eof: *INTEGER*
-- signals EOF
feature(*s*) **from** *STDC_CONSTANTS*
-- buffering
iofbf: *INTEGER*
-- full buffering
iolbf: *INTEGER*
-- line buffering
ionbf: *INTEGER*
-- no buffering
feature(*s*) **from** *STDC_CONSTANTS*
-- file positioning
seek_set: *INTEGER*
seek_cur: *INTEGER*
seek_end: *INTEGER*
feature(*s*) **from** *STDC_CONSTANTS*
-- Signal related constants
sig_dfl: *POINTER*
sig_err: *POINTER*
sig_ign: *POINTER*
feature(*s*) **from** *STDC_CONSTANTS*
-- Signals
sigabrt: *INTEGER*
sigfpe: *INTEGER*
-- erroneous arithmetic operation, such as zero divide or an
-- operation resulting in overflow
sigill: *INTEGER*
-- illegal instruction
sigint: *INTEGER*
-- receipt of an interactive attention signal
sigsegv: *INTEGER*

```
        -- invalid access to storage
    sigterm: INTEGER
feature(s) from STDC_CONSTANTS
    -- random numbers
    rand_max: INTEGER
        -- maximum value returned by the random function
feature(s) from STDC_CONSTANTS
    -- category constants
    lc_ctype: INTEGER
    lc_numeric: INTEGER
    lc_time: INTEGER
    lc_collate: INTEGER
    lc_monetary: INTEGER
    lc_all: INTEGER
feature(s) from STDC_CONSTANTS
    -- various
    clocks_per_sec: INTEGER
feature(s) from STDC_CONSTANTS
    -- exit codes
    exit_failure: INTEGER
        -- exit status when something has gone wrong
    exit_success: INTEGER
        -- exit status upon success
end of STDC_CONSTANTS
```

B.4 STDC_CURRENT_PROCESS

class *interface STDC_CURRENT_PROCESS*
feature(*s*) **from** *STDC_SECURITY_ACCESSOR*
-- The singleton, available to any because its used in preconditions
security: *STDC_SECURITY*
-- Singleton entry point for security.
feature(*s*) **from** *STDC_BASE*
-- errno
errno: *STDC_ERRNO*
feature(*s*) **from** *STDC_CURRENT_PROCESS*
-- my stdandard input/output/error
stdin: *STDC_TEXT_FILE*
stdout: *STDC_TEXT_FILE*
stderr: *STDC_TEXT_FILE*
invariant
accessing_real_singleton: *security_is_real_singleton*;
end *of STDC_CURRENT_PROCESS*

B.5 STDC_ENV_VAR

```
class interface STDC_ENV_VAR
creation
    make (a_name: STRING)
feature(s) from STDC_ENV_VAR
    -- Access
    exist: BOOLEAN
        -- Is this environment variable defined?
    name: STRING
        -- Name of environment variable.
    value: STRING
        -- Current value of environment variable.
invariant
    accessing_real_singleton: security_is_real_singleton;
end of STDC_ENV_VAR
```

B.6 STDC_FILE

STDC_FILE is a deferred class. Use *STDC_TEXT_FILE* for accessing and creating text files, or *STDC_BINARY_FILE* for binary files.

```
deferred class interface STDC_FILE
feature(s) from STDC_FILE
    -- Initialization
    create_read_write (path: STRING)
        -- Open file for update (reading and writing). If the file
        -- already exists, it is truncated to zero length.
        -- So permissions seem to remain.
    create_write (path: STRING)
        -- create new file for writing. If the file already exists,
        -- it is truncated to zero length.
        -- So permissions seem to remain.
    open (path, a_mode: STRING)
        -- open file in given mode
    open_append (path: STRING)
        -- append to exiting file or create file if it does not exist
    open_read (path: STRING)
        -- open file for reading
    open_read_write (path: STRING)
        -- Open file for reading and writing.
feature(s) from STDC_FILE
    -- Work with existing streams
    attach_to_stream (a_stream: POINTER; a_mode: STRING)
        -- Attach to a_stream. Does not become owner of stream so
        -- it will not close on close or when garbage collected.
feature(s) from STDC_FILE
    -- Reopen
    reopen (path, a_mode: STRING)
        -- Closes and then opens a stream.
feature(s) from STDC_FILE
    -- Control over buffering
    flush
        -- Updates this stream
    setbuf (buffer: POINTER)
        -- Determines how the stream will be buffered
        -- gives you a fully buffered input and output.
        -- Not sure: buffer should have at least BUFSIZ bytes?
        -- No operation should yet been performed on this file
        -- buffer = default_pointer: default buffer will be allocated
        -- buffer /= default_pointer implies buffer size = BUFSIZ
    set_buffer (buffer: POINTER)
        -- Determines how the stream will be buffered
        -- gives you a fully buffered input and output.
        -- Not sure: buffer should have at least BUFSIZ bytes?
```

```
        -- No operation should yet been performed on this file
        -- buffer = default_pointer: default buffer will be allocated
        -- buffer /= default_pointer implies buffer size = BUFSIZ
    set_full_buffering (buffer: POINTER; size: INTEGER)
        -- Determines buffering for a stream.
        -- If buffer is default_pointer, a buffer of size bytes
        -- will be allocated by this routine.
    set_line_buffering (buffer: POINTER; size: INTEGER)
        -- Determines buffering for a stream.
        -- Give NULL buffer so setvbuf will allocate a buffer.
    set_no_buffering
        -- Turn buffering off.
feature(s) from STDC_FILE
    -- read, C like
    last_byte: INTEGER
        -- last read character of get_character
        -- can be negative, so is more a last_shortint or so!
    getc
        -- Reads a C unsigned char and converts it to an integer,
        -- the result is left in last_byte
        -- This function probably can be used to read a single
        -- byte
    get_character
        -- Reads a C unsigned char and converts it to an integer,
        -- the result is left in last_byte
        -- This function probably can be used to read a single
        -- byte
    gets (bytes: INTEGER)
        -- Reads at most one less than bytes characters.
        -- No additional characters are read after a newline character
        -- or after end-of-file. If a newline character is read, it
        -- is returned too.
        -- Result is placed in last_string
    get_string (bytes: INTEGER)
        -- Reads at most one less than bytes characters.
        -- No additional characters are read after a newline character
        -- or after end-of-file. If a newline character is read, it
        -- is returned too.
        -- Result is placed in last_string
    read (buf: POINTER; offset, bytes: INTEGER)
        -- Read chunk, set last_read. offset determines how far
        -- in buf you want to start writing.
feature(s) from STDC_FILE
    -- write, C like
    putc (c: INTEGER)
        -- write a single character
    put_character (c: INTEGER)
```

```
        -- write a single character
    ungetc (c: INTEGER)
        -- pushes c back to the stream
        -- note that file positioning functions discard any
        -- pushed-back characters
    write (buf: POINTER; offset, bytes: INTEGER)
        -- write bytes bytes from buf at offset offset
        -- we do not really care if offset is positive or negative...
feature(s) from STDC_FILE
    -- read, Eiffel like
    last_read: INTEGER
        -- last read bytes by some read_XXXX or get_string call
    last_boolean: BOOLEAN
        -- last boolean read by read_boolean
    last_character: CHARACTER
        -- last character read by read_character
    last_double: DOUBLE
        -- last double lread by read_double
    last_integer: INTEGER
    last_real: REAL
        -- last real read by read_real
    last_string: STRING
        -- Last string read by read_string or
        -- get_string. Includes the end-of-line character, if any.
    read_boolean
        -- attempt to read back a boolean written by write_boolean
    read_buffer (buf: STDC_BUFFER; offset, bytes: INTEGER)
        -- More safe version of read in case you have a
        -- STDC_BUFFER object. Read starts at offset bytes in buf.
        -- Check last_read for number of bytes actually read.
    read_double
    read_character
        -- read a single character and set last_character
        -- if end-of-file encountered, eof is True
    read_integer
    read_real
    read_string (bytes: INTEGER)
        -- Read at most n characters, a value more expected by
        -- programmers not used to strings with a trailing byte.
        -- result is placed in last_string
        -- last_string includes the newline character if bytes
        -- are longer then the length of the actual line!
feature(s) from STDC_FILE
    -- write, Eiffel like
    last_written: INTEGER
        -- last written bytes by some write_XXXX call
    put (any: ANY)
```

-- write class as string
write_buffer (*buf*: *STDC_BUFFER*; *offset*, *bytes*: *INTEGER*)
-- more safe version of *write* in case you have a
-- STDC_BUFFER object
-- Check *last_written* for number of bytes actually written,
-- if you use asynchronous writing.
write_boolean (*b*: *BOOLEAN*)
-- write a boolean in Standard C %f format
write_character (*c*: *CHARACTER*)
-- write a single character
write_double (*d*: *DOUBLE*)
-- write a double in Standard C %f format
write_integer (*i*: *INTEGER*)
-- write an integer in Standard C %d format
write_real (*r*: *REAL*)
-- write a real in Standard C %f format
write_string (*s*: *STRING*)
-- write a string
puts (*s*: *STRING*)
-- write a string
put_string (*s*: *STRING*)
-- write a string
feature(*s*) **from** *STDC_FILE*
-- file position
getpos: *STDC_FILE_POSITION*
-- get the current position, use *set_position* to return to
-- this saved position
get_position: *STDC_FILE_POSITION*
-- get the current position, use *set_position* to return to
-- this saved position
rewind
-- Sets the file position to the beginning of the file
seek (*offset*: *INTEGER*)
-- set file position to given absolute *offset*
seek_from_current (*offset*: *INTEGER*)
-- set file position relative to current position
seek_from_end (*offset*: *INTEGER*)
-- set file position relative to end of file
setpos (*a_position*: *STDC_FILE_POSITION*)
-- set the current position
set_position (*a_position*: *STDC_FILE_POSITION*)
-- set the current position
tell: *INTEGER*
-- The current position
feature(*s*) **from** *STDC_FILE*
-- other
clearerr

```
        -- Clears end-of-file and error indicators for a stream.
    clear_error
        -- Clears end-of-file and error indicators for a stream.
feature(s) from STDC_FILE
    -- Access
    eof: BOOLEAN
        -- True if eof encountered by getc or,
        -- if the end-of-file indicator is set.
    error: BOOLEAN
        -- True if and only if the error indicator is set
    filename: STRING
        --The filename of this file.
    mode: STRING
        -- mode in which the file is opened/created
    resource_usage_can_be_increased: BOOLEAN
        -- Is it allowed to open another file?
feature(s) from STDC_FILE
    -- is mode binary or text
    is_binary_mode_specification (a_mode: STRING): BOOLEAN
        -- True if last character of a_mode = b
    is_text_mode_specification (a_mode: STRING): BOOLEAN
        -- True if last character of a_mode = t
feature(s) from STDC_FILE
    -- Low level handle functions
    do_close: BOOLEAN
        -- Close resource. Return False if an error occurred. Error
        -- value should be in errno. This routine may never call
        -- another object, else it cannot be used safely in
        -- dispose.
        -- This routine is usely redefined to actually close or
        -- deallocate the resource in addition of resetting handle.
    unassigned_value: POINTER
        -- The value that indicates that handle is unassigned.
invariant
    accessing_real_singleton: security_is_real_singleton;
    capacity_not_negative: capacity >= 0;
    valid_capacity: is_open = capacity > 0;
    open_implies_handle_assigned: is_open = stream /= unassigned_value;
    owned_implies_open: is_owner implies is_open;
    owned_implies_handle_assigned: is_owner implies stream /= unassigned_value;
    last_string_valid: last_string /= Void;
    gets_buf_valid: gets_buf /= Void;
end of deferred STDC_FILE
```

B.7 STDC_FILE_SYSTEM

```
class interface STDC_FILE_SYSTEM
feature(s) from STDC_FILE_SYSTEM
    -- path names
    expand_path (a_path: STRING): STDC_PATH
        -- returns a new path
feature(s) from STDC_FILE_SYSTEM
    -- rename files/directories, remove files/directories
    remove_file (a_path: STRING)
        -- Removes a file from a directory.
        -- For Standard C, its implementation defined what
        -- remove_file does if file is opened by some process
        -- (remove_file fails on Windows for example).
        -- doesnt remove a directory.
    rename_to (current_path, new_path: STRING)
        -- Rename a file or a directory.
        -- new_path should not be an existing path.
feature(s) from STDC_FILE_SYSTEM
    -- accessibility of files
    is_modifiable (a_path: STRING): BOOLEAN
        -- Is a_path readable and writable by this program?
        -- Does this by attemting to open a_path file read/write.
    is_readable (a_path: STRING): BOOLEAN
        -- tests if file is readable by this program
        -- does this by attemting to open a_path file read-only
invariant
    accessing_real_singleton: security_is_real_singleton;
end of STDC_FILE_SYSTEM
```

B.8 STDC_SECURITY

```
class interface STDC_SECURITY
feature(s) from STDC_SECURITY
    -- modes
    make_allow_all
        -- just allow everything
    make_allow_sandbox
        -- allow very little, use for setuid root programs
feature(s) from STDC_SECURITY
    -- the security aspects
    cpu: STDC_SECURITY_CPU
    error_handling: STDC_SECURITY_ERROR_HANDLING
    files: STDC_SECURITY_FILES
    memory: STDC_SECURITY_MEMORY
feature(s) from STDC_SECURITY
    -- various
    assert_once_memory_allocated
        -- make sure that certain once functions in STDC_BASE are
        -- called. These once functions are called when an error
        -- occurs, at that time there might not be memory left to
        -- create them
invariant
    accessing_real_singleton: security_is_real_singleton;
    remain_single: Current = the_singleton;
end of STDC_SECURITY
```

B.9 STDC_SIGNAL

class *interface STDC_SIGNAL*
creation
    make (*a_value*: *INTEGER*)
feature(*s*) **from** *STDC_SIGNAL*
    -- creation
    make (*a_value*: *INTEGER*)
feature(*s*) **from** *STDC_SIGNAL*
    -- set signal properties, make effective with *apply*
    apply
        -- Make changes effective.
    set_default_action
        -- install signal-specific default action
    set_ignore_action
        -- ignore signal
    set_handler (*a_handler*: *STDC_SIGNAL_HANDLER*)
        -- Install ones own signal handler.
feature(*s*) **from** *STDC_SIGNAL*
    -- signal functions
    raise
        -- raise the signal
feature(*s*) **from** *STDC_SIGNAL*
    -- signal state
    is_ignorable: *BOOLEAN*
        -- All signals Standard C knows about are ignorable...
    value: *INTEGER*
        -- the signal
invariant
    accessing_real_singleton: *signal_switch_is_real_singleton*;
    accessing_real_singleton: *security_is_real_singleton*;
    valid_signal_value: *value >= 1*;
end *of STDC_SIGNAL*

B.10 STDC_SIGNAL_HANDLER

deferred class *interface STDC_SIGNAL_HANDLER*
invariant
accessing_real_singleton: *signal_switch_is_real_singleton*;
end *of* **deferred** *STDC_SIGNAL_HANDLER*

B.11 STDC_SYSTEM

class *interface STDC_SYSTEM*
feature(*s*) **from** *STDC_SYSTEM*
-- run-time determined queries
is_shell_available: *BOOLEAN*
-- Return True if command interpreter is available
feature(*s*) **from** *STDC_SYSTEM*
-- compile time determined queries
clocks_per_second: *INTEGER*
-- number per second of the value returned by the *clock* function
feature(*s*) **from** *STDC_SYSTEM*
-- endianess
is_big_endian: *BOOLEAN*
-- True if this is a big endian architecture
is_little_endian: *BOOLEAN*
-- True if this is a little endian architecture
invariant
accessing_real_singleton: *security_is_real_singleton*;
end *of STDC_SYSTEM*

B.12 STDC_TIME

```
class interface STDC_TIME
creation
    make_date (a_year, a_month, a_day: INTEGER)
        -- create a time according to this day, time 00:00:00
        -- date is assumed to be local date
    make_date_time (a_year, a_month, a_day, a_hour, a_minute, a_second: INTEGER)
        -- date is assumed to be local date
        -- we assume daylight saving time setting in effect is
        -- available from system
    make_from_now
        -- Make value equal to current unix time.
        -- Afterwards call to_local or to_utc to turn individual
        -- fields in local time or in utc time.
    make_time (a_hour, a_minute, a_second: INTEGER)
        -- We assume daylight saving time setting in effect is
        -- available from system.
        -- Day will be January 1, 1970
    make_from_unix_time (a_value: INTEGER)
        -- a_value is a time_t value.
        -- Afterwards call to_local or to_utc to turn individual
        -- fields in local time or in utc time.
feature(s) from STDC_TIME
    -- Initialization
    make_date (a_year, a_month, a_day: INTEGER)
        -- create a time according to this day, time 00:00:00
        -- date is assumed to be local date
    make_date_time (a_year, a_month, a_day, a_hour, a_minute, a_second: INTEGER)
        -- date is assumed to be local date
        -- we assume daylight saving time setting in effect is
        -- available from system
    make_from_now
        -- Make value equal to current unix time.
        -- Afterwards call to_local or to_utc to turn individual
        -- fields in local time or in utc time.
    make_time (a_hour, a_minute, a_second: INTEGER)
        -- We assume daylight saving time setting in effect is
        -- available from system.
        -- Day will be January 1, 1970
    make_from_unix_time (a_value: INTEGER)
        -- a_value is a time_t value.
        -- Afterwards call to_local or to_utc to turn individual
        -- fields in local time or in utc time.
feature(s) from STDC_TIME
    -- make individual time fields valid
    to_local
```

-- switch time fields to local time
to_utc
-- switch time fields to utc time
feature(*s*) **from** *STDC_TIME*
-- manually set individual time fields
set_date (*a_year, a_month, a_day*: *INTEGER*)
-- set date part, time remains unchanged
set_date_time (*a_year, a_month, a_day, a_hour, a_minute, a_second*: *INTEGER*)
-- we assume daylight saving time setting in effect (or not)
-- has been set
set_dst_to_current
-- Let system figure out if daylight saving time is in effect.
set_dst_to_none
-- Daylight saving time is not in effect.
set_dst_in_effect
-- Daylight saving time is in effect.
set_time (*a_hour, a_minute, a_second*: *INTEGER*)
-- set time part, date remains unchanged
feature(*s*) **from** *STDC_TIME*
-- individual time fields, need call to *to_local* or *to_utc*
year: *INTEGER*
month: *INTEGER*
day: *INTEGER*
-- Day of the month
weekday: *INTEGER*
-- Days since Sunday
day_of_year: *INTEGER*
-- Days since January 1st
hour: *INTEGER*
minute: *INTEGER*
second: *INTEGER*
is_daylight_savings_in_effect: *BOOLEAN*
-- Return True if we know for sure daylight savings is effective
is_daylight_savings_unknown: *BOOLEAN*
-- Return True if we do not know if daylight savings is effective
feature(*s*) **from** *STDC_TIME*
-- time as string
short_weekday_name: *STRING*
-- Abbreviated weekday name
weekday_name: *STRING*
-- Full weekday name
short_month_name: *STRING*
-- Abbreviated month name
month_name: *STRING*
-- Full month name
format (*format_str*: *STRING*): *STRING*
-- Return formatted date time according to *format_str*. See

```
        -- man strftime for details.
    default_format: STRING
        -- Returns a string of the form "Mon Apr 17 21:49:20 2000"
    local_date_string: STRING
        -- Return date part in format local to current country
    local_time_string: STRING
        -- Return time part in format local to current country
feature(s) from STDC_TIME
    -- date calculations
    is_equal (other: like Current): BOOLEAN
        -- Is other attached to an object considered equal to
        -- current object ?
    infix "-" (other: like Current): like Current
        -- Creates a new time which is the difference between
        -- Current and Other
    infix "<" (other: like Current): BOOLEAN
        -- Is current object less than other?
feature(s) from STDC_TIME
    -- state
    hash_code: INTEGER
        -- The hash-code value of Current.
    value: INTEGER
        -- time in seconds from January 1, 1970
        -- perhaps since 1980 for Windows systems
feature(s) from STDC_TIME
    -- non POSIX, but Gates specific stuff
    minimum_year: INTEGER
        -- returns the minimum year for the current platform
        -- for POSIX is 1970, for Windows is 1980
invariant
    accessing_real_singleton: security_is_real_singleton;
    valid_tm_struct: tm /= Void;
end of STDC_TIME
```

In this chapter:

C Short listing of abstract classes

An abstract class is somewhat above the Standard C classes, and between the features you get when you use a POSIX or Windows class. It is mainly aimed at users who want to write software usable on Unix and Windows, and who do not want to use a POSIX emulator.

You never use an abstract class directly, always use the corresponding effective EPX_XXXX, for which there is a variant in the `src/posix` or `src/windows` directory.

C.1 *ABSTRACT_CURRENT_PROCESS*

deferred class *interface ABSTRACT_CURRENT_PROCESS*
feature(*s*) **from** *STDC_SECURITY_ACCESSOR*
-- The singleton, available to any because its used in preconditions
security: *STDC_SECURITY*
-- Singleton entry point for security.
feature(*s*) **from** *STDC_BASE*
-- errno
errno: *STDC_ERRNO*
feature(*s*) **from** *STDC_CURRENT_PROCESS*
-- my stdandard input/output/error
stdin: *STDC_TEXT_FILE*
stdout: *STDC_TEXT_FILE*
stderr: *STDC_TEXT_FILE*
feature(*s*) **from** *ABSTRACT_PROCESS*
-- Process properties
pid: *INTEGER*
-- The process identifier.
is_pid_valid: *BOOLEAN*
-- current process id is always valid
feature(*s*) **from** *ABSTRACT_PROCESS*
-- Signal this process
terminate
-- Attempt to gracefully terminate this process.
require
valid_pid: *is_pid_valid*
feature(*s*) **from** *ABSTRACT_CURRENT_PROCESS*
-- every process also has standard file descriptors which might not be compatible with stdin/stdout/stderr (Wind
fd_stdin: *ABSTRACT_FILE_DESCRIPTOR*
fd_stdout: *ABSTRACT_FILE_DESCRIPTOR*

```
    fd_stderr: ABSTRACT_FILE_DESCRIPTOR
invariant
    accessing_real_singleton: security_is_real_singleton;
end of deferred ABSTRACT_CURRENT_PROCESS
```

C.2 ABSTRACT_EXEC_PROCESS

```
deferred class interface ABSTRACT_EXEC_PROCESS
feature(s) from ABSTRACT_EXEC_PROCESS
    -- Initialization
    make (a_program: STRING; a_arguments: ARRAY[STRING])
    make_capture_input (a_program: STRING; a_arguments: ARRAY[STRING])
    make_capture_output (a_program: STRING; a_arguments: ARRAY[STRING])
    make_capture_io (a_program: STRING; a_arguments: ARRAY[STRING])
        -- Why not use threedirectional i/o, because youre getting
        -- yourself in great, great trouble anyway.
        -- A bit of advice: call stdin.close before starting to call
        -- stdout.read_string and such...
    make_capture_all (a_program: STRING; a_arguments: ARRAY[STRING])
        -- Threedirectional i/o is a great way to get yourself in trouble.
feature(s) from ABSTRACT_EXEC_PROCESS
    -- (re)set arguments
    set_arguments (a_arguments: ARRAY[STRING])
feature(s) from ABSTRACT_EXEC_PROCESS
    -- i/o capturing
    capture_input: BOOLEAN
        -- is input captured on execute?
    capture_output: BOOLEAN
        -- is output captured on execute?
    capture_error: BOOLEAN
        -- is error captured on execute?
    set_capture_input (on: BOOLEAN)
    set_capture_output (on: BOOLEAN)
    set_capture_error (on: BOOLEAN)
    fd_stdin: ABSTRACT_FILE_DESCRIPTOR
    fd_stdout: ABSTRACT_FILE_DESCRIPTOR
    fd_stderr: ABSTRACT_FILE_DESCRIPTOR
feature(s) from ABSTRACT_EXEC_PROCESS
    -- Execute
    execute
        -- Executes program_name. After execution, at some point in
        -- time, you have to wait or wait_for for this process to
        -- terminate.
        require
            not_already_started: is_terminated
feature(s) from ABSTRACT_EXEC_PROCESS
    -- Actions that parent may execute
    wait_for (suspend: BOOLEAN)
        -- Wait for child process to terminate if suspend.
        require
            pid_refers_to_child: is_pid_valid;
            not_terminated: not is_terminated
```

```
        ensure
            stdin_closed: is_terminated implies fd_stdin = Void or else fd_stdin.is_closed;
            stdout_closed: is_terminated implies fd_stdout = Void or else fd_stdout.is_closed;
            stderr_closed: is_terminated implies fd_stderr = Void or else fd_stderr.is_closed;
            terminated: suspend implies is_terminated
feature(s) from ABSTRACT_EXEC_PROCESS
    -- Accessible state
    program_name: STDC_PATH
        -- program to execute
    arguments: ARRAY[STRING]
        -- arguments to pass to program
invariant
    accessing_real_singleton: security_is_real_singleton;
    program_name_not_empty: program_name /= Void and then not program_name.is_empty;
    arguments_not_void: arguments /= Void;
end of deferred ABSTRACT_EXEC_PROCESS
```

C.3 ABSTRACT_FILE_DESCRIPTOR

```
deferred class interface ABSTRACT_FILE_DESCRIPTOR
feature(s) from STDC_SECURITY_ACCESSOR
    -- The singleton, available to any because its used in preconditions
    security: STDC_SECURITY
        -- Singleton entry point for security.
feature(s) from STDC_BASE
    -- errno
    errno: STDC_ERRNO
feature(s) from ABSTRACT_FILE_DESCRIPTOR
    -- Creation
    open (a_path: STRING; a_flags: INTEGER)
        -- open given file with access given by flags
    open_read (a_path: STRING)
        -- open given file with read-only access
    open_write (a_path: STRING)
    open_read_write (a_path: STRING)
    open_truncate (a_path: STRING)
    create_read_write (a_path: STRING)
        -- Always create a file, existing or not.
        -- Give read/write permissions to user only.
    create_write (a_path: STRING)
        -- Always create a file, existing or not.
        -- Give read/write permissions to user only.
    create_with_mode (a_path: STRING; flags, mode: INTEGER)
        -- create a file according to flags and with mode access
        -- permissions. Make sure you have th O_CREAT flag in flags
        -- if you really want to create something!
feature(s) from ABSTRACT_FILE_DESCRIPTOR
    -- Special creation
    attach_to_fd (a_fd: INTEGER)
        -- Create file descriptor with value a_fd. File descriptor
        -- does not become owner, so it will not close a_fd.
    make_as_duplicate (another: ABSTRACT_FILE_DESCRIPTOR)
        -- On creation, create a duplicate from another file descriptor
        -- As normal call, closes its own descriptor first (if open) and
        -- duplicates next.
feature(s) from ABSTRACT_FILE_DESCRIPTOR
    -- Close
    close
        -- We always describe an existing object, however user
        -- probably wants to have control about closing a file.
    detach
        -- Forget the current file descriptor.
feature(s) from ABSTRACT_FILE_DESCRIPTOR
    -- Change ownership of the descriptor. Can help to influence subtile garbage collector problems
```

make_owner
-- this file descriptor will (start to) own its descriptor
unown
-- When a stream is opened on a file descriptor the file
-- descriptor itself should not close itself, the stream
-- will close it.
feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*
-- Stream or file behaviour
is_streaming: *BOOLEAN*
-- Is data from this file descriptor coming through a network
-- stream?
set_streaming (*enable*: *BOOLEAN*)
-- Influence behaviour of certain functions if they should be
-- optimized for data coming from disk or data coming from
-- the network. In particular *is_streaming* implies that a
-- client application is prepared to handle *read*s that
-- return less than the requested number of bytes, but dont
-- assume that means end-of-file.
feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*
-- Raw read and write
last_blocked: *BOOLEAN*
-- True if last read or write call would be blocked
last_read: *INTEGER*
-- How many bytes were read by last call to *read*
-- -1 implies last_blocked
last_written: *INTEGER*
-- How many bytes were written by last call to write
-- -1 implies last_blocked
read_loop_disabled: *BOOLEAN*
-- For data coming from the network, a read does not always
-- return the requested number of bytes. In such a case,
-- reading again probably will return more bytes as they have
-- arrived in the network buffers, or perhaps by reading you
-- have freed the network buffers for more data.
-- However, certain file descriptors block when reading
-- again, because they dont return EOF when there is no more
-- data. A typical example is a character special file.
-- And sometimes the application can handle streaming reads
-- just fine and doesnt care if not all requested bytes are
-- returned.
-- This variable influences if *read* will attempt to read
-- more bytes in its loop, or not.
read (*buf*: *POINTER*; *offset*, *nbytes*: *INTEGER*)
-- Read data into *buf* at *offset* for *nbytes* bytes.
-- Number of bytes actually read are available in last_read.
-- Dont mix this routine with read_string or read_character!
write (*buf*: *POINTER*; *offset*, *nbytes*: *INTEGER*)

```
        -- write given data from buf at offset, for nbytes bytes.
feature(s) from ABSTRACT_FILE_DESCRIPTOR
    -- Safer read/write
    read_buffer (buf: STDC_BUFFER; offset, nbytes: INTEGER)
        -- more safe version of read in case you have a
        -- STDC_BUFFER object
    write_buffer (buf: STDC_BUFFER; offset, bytes: INTEGER)
        -- more safe version of write in case you have a
        -- STDC_BUFFER object
feature(s) from ABSTRACT_FILE_DESCRIPTOR
    -- Read routines that can be mixed with read calls
    last_line: STRING
        -- last line read by read_line (includes %N), see STRING_HELPER.chop
    read_line (max_length: INTEGER)
        -- Raw, and slow, read of characters up to end of line. Can
        -- be safely mixed with read, unlike read_string. Only
        -- max_length characters are returned.
feature(s) from ABSTRACT_FILE_DESCRIPTOR
    -- Eiffel like output
    put (a: ANY)
        -- write any Eiffel object as string
    write_character (c: CHARACTER)
    write_string (s: STRING)
    puts (s: STRING)
    put_string (s: STRING)
feature(s) from ABSTRACT_FILE_DESCRIPTOR
    -- Buffered input, line reading instead of block reading, dont mix with raw read calls!
    last_character: CHARACTER
    last_string: STRING
        -- last read string (includes %N), see STRING_HELPER.chop
    read_character
        -- Sets last_character.
    read_string (a_size: INTEGER)
        -- Implements line reading on top of read. Sets
        -- last_string which includes the new line character if
        -- any. Reads until eof or new line encountered, returns max
        -- a_size characters.
feature(s) from ABSTRACT_FILE_DESCRIPTOR
    -- file position
    seek (offset: INTEGER)
        -- set file position to given absolute offset
    seek_from_current (offset: INTEGER)
        -- set file position relative to current position
    seek_from_end (offset: INTEGER)
        -- set file position relative to end of file
feature(s) from ABSTRACT_FILE_DESCRIPTOR
    -- queries
```

eof: *BOOLEAN*
-- True if end-of-file reached.
-- Currently Im unsure if detection is reliable
isatty: *BOOLEAN*
-- return true if handle associated with character device
is_attached_to_terminal: *BOOLEAN*
-- return true if handle associated with character device
is_blocking_io: *BOOLEAN*
-- True if blocking i/o enabled (default)
is_closed: *BOOLEAN*
-- Is file descriptor is closed?
is_open: *BOOLEAN*
-- Does *value* describes a valid file descriptor?
is_owner: *BOOLEAN*
-- Does this file descriptor own its descriptor? Only when
-- it owns the descriptor it will close it when *close* or
-- *dispose* is called
status: *ABSTRACT_STATUS*
-- The status for this file descriptor. Cached value,
-- refreshed only when file reopened.
value: *INTEGER*
-- return the value of the file descriptor
feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*
-- Access
fd: *INTEGER*
-- The actual file descriptor.
invariant
accessing_real_singleton: *security_is_real_singleton*;
valid_internal_file_descriptor: *fd >= -1*;
valid_open: *is_open* **implies** *fd >= 0*;
valid_close: **not** *is_open* **implies** *fd = -1*;
valid_status: *is_closed* **implies** *my_status = Void*;
open_and_close_in_balance: *is_open* = **not** *is_closed*;
owner_implies_open: *is_owner* **implies** *is_open*;
end *of* **deferred** *ABSTRACT_FILE_DESCRIPTOR*

C.4 ABSTRACT_FILE_SYSTEM

```
deferred class interface ABSTRACT_FILE_SYSTEM
feature(s) from STDC_SECURITY_ACCESSOR
    -- The singleton, available to any because its used in preconditions
    security: STDC_SECURITY
        -- Singleton entry point for security.
feature(s) from STDC_BASE
    -- errno
    errno: STDC_ERRNO
feature(s) from STDC_FILE_SYSTEM
    -- path names
    expand_path (a_path: STRING): STDC_PATH
        -- returns a new path
feature(s) from STDC_FILE_SYSTEM
    -- rename files/directories, remove files/directories
    remove_file (a_path: STRING)
        -- Removes a file from a directory.
        -- For Standard C, its implementation defined what
        -- remove_file does if file is opened by some process
        -- (remove_file fails on Windows for example).
        -- doesnt remove a directory.
    rename_to (current_path, new_path: STRING)
        -- Rename a file or a directory.
        -- new_path should not be an existing path.
feature(s) from STDC_FILE_SYSTEM
    -- accessibility of files
    is_modifiable (a_path: STRING): BOOLEAN
        -- tests if file is readable and writable by this program
        -- uses real user ID and real group ID instead of effective ones
    is_readable (a_path: STRING): BOOLEAN
        -- tests if file is readable by this program
        -- uses real user ID and real group ID instead of effective ones
feature(s) from ABSTRACT_FILE_SYSTEM
    -- directory access
    change_directory (a_directory: STRING)
        -- Changes the current working directory
    chdir (a_directory: STRING)
        -- Changes the current working directory
    current_directory: STRING
        -- The current directory
    getcwd: STRING
        -- The current directory
    pwd: STRING
        -- The current directory
    make_directory (a_directory: STRING)
        -- Makes a directory, only accessible by owner
```

```
    mkdir (a_directory: STRING)
        -- Makes a directory, only accessible by owner
    remove_directory (a_directory: STRING)
        -- Removes an empty directory, see also force_remove_directory
    rmdir (a_directory: STRING)
        -- Removes an empty directory, see also force_remove_directory
    force_remove_directory (a_directory: STRING)
        -- Removes a directory, even when not empty.
        -- I suggest you do not have hard or symbolic links in a_directory...
feature(s) from ABSTRACT_FILE_SYSTEM
    -- file statistics
    status (a_path: STRING): ABSTRACT_STATUS_PATH
        -- Gets information about a file
        require
            valid_path: a_path /= Void and then not a_path.is_empty;
            existing_file: is_existing(a_path)
        ensure
            status_returned: Result /= Void
feature(s) from ABSTRACT_FILE_SYSTEM
    -- directory browsing
    browse_directory (a_path: STRING): ABSTRACT_DIRECTORY
        -- Gets information about a directory
        require
            valid_path: a_path /= Void and then not a_path.is_empty;
            path_is_directory: security.error_handling.exceptions_enabled and then status(a_path).is_directory
        ensure
            directory_returned: Result /= Void
feature(s) from ABSTRACT_FILE_SYSTEM
    -- accessibility of files
    last_access_result: INTEGER
        -- value of last access test
    is_accessible (a_path: STRING; a_mode: INTEGER): BOOLEAN
        -- Tests for file accessibility
    access (a_path: STRING; a_mode: INTEGER): BOOLEAN
        -- Tests for file accessibility
    is_directory (a_path: STRING): BOOLEAN
        -- return True if a_path exists and if it is a directory
    is_existing (a_path: STRING): BOOLEAN
        -- tests if file does exist, not if it is readable or writable by
        -- this program!
        -- uses real user ID and real group ID instead of effective ones
    is_empty (a_path: STRING): BOOLEAN
        -- True if file exists and has a size equal to zero.
    is_executable (a_path: STRING): BOOLEAN
        -- tests if file is executable by this program
    is_writable (a_path: STRING): BOOLEAN
        -- tests if file is writable by this program
```

```
        -- uses real user ID and real group ID instead of effective ones
feature(s) from ABSTRACT_FILE_SYSTEM
    -- various
    is_case_sensitive: BOOLEAN
        -- is file system case sensitive or not?
        -- This query is dedicated to jwz
    path_separator: CHARACTER
        -- What is the path separator?
feature(s) from ABSTRACT_FILE_SYSTEM
    -- file system properties
    temporary_directory: STRING
        -- returns temporary directory.
        ensure
            directory_returned: Result /= Void;
            directory_exists: is_directory(Result);
            directory_is_writable: is_modifiable(Result);
            last_char_not_separator: Result.item(Result.count) /= path_separator
invariant
    accessing_real_singleton: security_is_real_singleton;
end of deferred ABSTRACT_FILE_SYSTEM
```

C.5 ABSTRACT_PIPE

deferred class *interface ABSTRACT_PIPE*
feature(*s*) **from** *ABSTRACT_PIPE*
    -- creation
    make
feature(*s*) **from** *ABSTRACT_PIPE*
    -- pipe operations
    close
feature(*s*) **from** *ABSTRACT_PIPE*
    -- the pipe
    fdout: *ABSTRACT_FILE_DESCRIPTOR*
    fdin: *ABSTRACT_FILE_DESCRIPTOR*
invariant
    accessing_real_singleton: *security_is_real_singleton*;
    valid_pipe: *fdin* /= *Void* **and** *fdout* /= *Void*;
end *of* **deferred** *ABSTRACT_PIPE*

C.6 ABSTRACT_STATUS

```
deferred class interface ABSTRACT_STATUS
feature(s) from ABSTRACT_STATUS
    refresh
        -- refresh the cached information
feature(s) from ABSTRACT_STATUS
    -- stat members
    atime: INTEGER
        -- Unix time of last access.
    access_time: INTEGER
        -- Unix time of last access.
    device_number: INTEGER
        -- ID of device containing the file.
        -- Windows: Drive number of the disk containing the file.
    is_character_special: BOOLEAN
        -- Is this file a character-special file?
    is_directory: BOOLEAN
    is_fifo: BOOLEAN
    is_regular_file: BOOLEAN
    mtime: INTEGER
        -- Unix time of last data modification.
    modification_time: INTEGER
        -- Unix time of last data modification.
    nlink: INTEGER
    number_of_hard_links: INTEGER
    size: INTEGER
        -- Size of file in bytes.
    status_change_time: INTEGER
        -- Unix time of last status change.
        -- For example changing the permission bits will set this time.
feature(s) from ABSTRACT_STATUS
    -- Direct access to the individual stat fields, not recommended
    unix_mode: INTEGER
invariant
    accessing_real_singleton: security_is_real_singleton;
    valid_stat: stat /= Void and then stat.capacity >= abstract_stat_size;
end of deferred ABSTRACT_STATUS
```

In this chapter:

D Short (flat) listing of POSIX classes

D.1 POSIX_ASYNC_IO_REQUEST

```
class interface POSIX_ASYNC_IO_REQUEST
creation
    make (a_fd: POSIX_FILE_DESCRIPTOR)
feature(s) from POSIX_ASYNC_IO_REQUEST
    -- creation
    make (a_fd: POSIX_FILE_DESCRIPTOR)
feature(s) from POSIX_ASYNC_IO_REQUEST
    -- request properties
    raw_pointer: POINTER
        -- Location for read or written data, usually buffer is a
        -- better idea.
    count: INTEGER
        -- number of bytes to read/write
    offset: INTEGER
        -- file offset
feature(s) from POSIX_ASYNC_IO_REQUEST
    -- set request properties
    set_buffer (a_buffer: STDC_BUFFER)
        -- set memory location to read/write from.
    set_count (a_count: INTEGER)
        -- set number of bytes to read/write
    set_offset (a_offset: INTEGER)
    set_raw_pointer (a_pointer: POINTER)
        -- set memory location to read/write from. Make sure you have
        -- called set_count first!
feature(s) from POSIX_ASYNC_IO_REQUEST
    -- basic read/write requests
    read
        -- execute async read request
    write
        -- execute async write request
feature(s) from POSIX_ASYNC_IO_REQUEST
    -- Eiffel friendly reads and writes
```

```
    last_string: STRING
        -- attempt to return buffer as an Eiffel string
        -- buffer should have a terminating byte!
    read_string
    write_string (text: STRING)
feature(s) from POSIX_ASYNC_IO_REQUEST
    -- other operations
    cancel_failed: BOOLEAN
        -- set by cancel, True if cancel request failed, probably
        -- because operation was already performed
    cancel
        -- cancel request
    synchronize
        -- force all i/o operations queued for the file descriptor
        -- associated with this request to the synchronous state.
        -- Function returns when the request has been initiated or
        -- queued to the file or device (even when the data cannot be
        -- synchronized immediately)
    synchronize_data
        -- force all i/o operations queued for the file descriptor
        -- associated with this request to the synchronous state.
        -- Function returns when the request has been initiated or
        -- queued to the file or device (even when the data cannot be
        -- synchronized immediately)
    wait_for
        -- suspend process, until request completed
feature(s) from POSIX_ASYNC_IO_REQUEST
    -- state
    buffer: STDC_BUFFER
        -- buffer where data that is being read/write comes from,
        -- unless set_pointer has been called
    fd: POSIX_FILE_DESCRIPTOR
    is_pending: BOOLEAN
        -- True if io request is still pending
    return_status: INTEGER
        -- return status of asynchronous i/o operation, equal to what
        -- the synchronous read, write of fsync would have returned
invariant
    accessing_real_singleton: security_is_real_singleton;
    valid_aiocb: aiocb /= Void;
    synced_buffer_and_raw_pointer: buffer /= Void implies buffer.ptr = raw_pointer;
end of POSIX_ASYNC_IO_REQUEST
```

D.2 POSIX_BASE

class *interface POSIX_BASE*
invariant
accessing_real_singleton: *security_is_real_singleton*;
end *of POSIX_BASE*

D.3 POSIX_CHILD_PROCESS

```
deferred class interface POSIX_CHILD_PROCESS
feature(s) from POSIX_CHILD_PROCESS
    -- Childs pid
    pid: INTEGER
        -- The process identifier.
    is_pid_valid: BOOLEAN
        -- return True if this object refers to a child process, so
        -- it has an id
feature(s) from POSIX_CHILD_PROCESS
    -- Actions that parent may execute
    wait_for (suspend: BOOLEAN)
        -- wait for this process to terminate. If suspend then we
        -- wait until the information about this process is available,
        -- else we return immediately.
        -- If suspend is False, check the running property to see
        -- if this child is really terminated.
invariant
    accessing_real_singleton: security_is_real_singleton;
end of deferred POSIX_CHILD_PROCESS
```

D.4 POSIX_CONSTANTS

class *interface POSIX_CONSTANTS*
feature(*s*) **from** *STDC_CONSTANTS*
-- error codes
edom: *INTEGER*
-- Math argument out of domain of function
erange: *INTEGER*
-- Math result not representable
feature(*s*) **from** *STDC_CONSTANTS*
-- standard streams
stream_stdin: *POINTER*
stream_stdout: *POINTER*
stream_stderr: *POINTER*
feature(*s*) **from** *STDC_CONSTANTS*
-- characters
const_eof: *INTEGER*
-- signals EOF
feature(*s*) **from** *STDC_CONSTANTS*
-- buffering
iofbf: *INTEGER*
-- full buffering
iolbf: *INTEGER*
-- line buffering
ionbf: *INTEGER*
-- no buffering
feature(*s*) **from** *STDC_CONSTANTS*
-- file positioning
seek_set: *INTEGER*
seek_cur: *INTEGER*
seek_end: *INTEGER*
feature(*s*) **from** *STDC_CONSTANTS*
-- Signal related constants
sig_dfl: *POINTER*
sig_err: *POINTER*
sig_ign: *POINTER*
feature(*s*) **from** *STDC_CONSTANTS*
-- Signals
sigabrt: *INTEGER*
sigfpe: *INTEGER*
-- erroneous arithmetic operation, such as zero divide or an
-- operation resulting in overflow
sigill: *INTEGER*
-- illegal instruction
sigint: *INTEGER*
-- receipt of an interactive attention signal
sigsegv: *INTEGER*

-- invalid access to storage
sigterm: *INTEGER*
feature(*s*) **from** *STDC_CONSTANTS*
-- random numbers
rand_max: *INTEGER*
-- maximum value returned by the *random* function
feature(*s*) **from** *STDC_CONSTANTS*
-- category constants
lc_ctype: *INTEGER*
lc_numeric: *INTEGER*
lc_time: *INTEGER*
lc_collate: *INTEGER*
lc_monetary: *INTEGER*
lc_all: *INTEGER*
feature(*s*) **from** *STDC_CONSTANTS*
-- various
clocks_per_sec: *INTEGER*
feature(*s*) **from** *STDC_CONSTANTS*
-- exit codes
exit_failure: *INTEGER*
-- exit status when something has gone wrong
exit_success: *INTEGER*
-- exit status upon success
feature(*s*) **from** *POSIX_CONSTANTS*
-- error codes
eagain: *INTEGER*
ebadf: *INTEGER*
eexist: *INTEGER*
einprogress: *INTEGER*
eintr: *INTEGER*
enoent: *INTEGER*
-- A file or directory does not exist
enospc: *INTEGER*
-- There is no free space remaining on the device
enosys: *INTEGER*
feature(*s*) **from** *POSIX_CONSTANTS*
-- standard file numbers
stderr_fileno: *INTEGER*
stdin_fileno: *INTEGER*
stdout_fileno: *INTEGER*
feature(*s*) **from** *POSIX_CONSTANTS*
-- posix open symbolic constants
o_append: *INTEGER*
-- Set the file offset to the end-of-file prior to each write
o_creat: *INTEGER*
-- If the file does not exist, allow it to be created. This
-- flag indicates that the mode argument is present in the

-- call to open.
o_dsync: *INTEGER*
-- Write according to synchronized i/o data integrity completion
o_excl: *INTEGER*
-- Open fails if the file already exists
o_exclusive: *INTEGER*
-- Open fails if the file already exists
o_noctty: *INTEGER*
-- prevents terminal from becoming the controlling terminal
-- for this process
o_nonblock: *INTEGER*
-- Do not wait for device or file to be ready or available
o_rdonly: *INTEGER*
-- Open for reading only
o_rdwr: *INTEGER*
-- Open fo reading and writing
o_rsync: *INTEGER*
-- Synchronized read i/o operations
o_sync: *INTEGER*
-- Write according to synchronized i/o file integrity completion
o_trunc: *INTEGER*
-- Use only on ordinary files opened for writing. It causes
-- the file to be truncated to zero length.
o_wronly: *INTEGER*
-- Open for writing only
feature(*s*) **from** *POSIX_CONSTANTS*
-- posix permission symbolic constants
s_irusr: *INTEGER*
s_iread: *INTEGER*
s_iwusr: *INTEGER*
s_iwrite: *INTEGER*
s_ixusr: *INTEGER*
s_iexec: *INTEGER*
s_irgrp: *INTEGER*
s_iwgrp: *INTEGER*
s_ixgrp: *INTEGER*
s_iroth: *INTEGER*
s_iwoth: *INTEGER*
s_ixoth: *INTEGER*
s_isuid: *INTEGER*
s_isgid: *INTEGER*
feature(*s*) **from** *POSIX_CONSTANTS*
-- Posix accessibility constants
f_ok: *INTEGER*
r_ok: *INTEGER*
w_ok: *INTEGER*
x_ok: *INTEGER*

feature(*s*) **from** *POSIX_CONSTANTS*
-- Posix signal constants
sa_nocldstop: *INTEGER*
sighup: *INTEGER*
-- hangup detected on controlling terminal or death of
-- controlling process
signal_hangup: *INTEGER*
-- hangup detected on controlling terminal or death of
-- controlling process
sigalrm: *INTEGER*
-- Timeout signal, such as initiated by the alarm() function
-- or see POSIX_TIMED_COMMAND
signal_alarm: *INTEGER*
-- Timeout signal, such as initiated by the alarm() function
-- or see POSIX_TIMED_COMMAND
sigchld: *INTEGER*
-- Child process terminated or stopped
signal_child: *INTEGER*
-- Child process terminated or stopped
sigkill: *INTEGER*
-- Termination signal (cannot be caught or ignored)
signal_kill: *INTEGER*
-- Termination signal (cannot be caught or ignored)
sigpipe: *INTEGER*
-- Write on a pipe with no readers
signal_pipe: *INTEGER*
-- Write on a pipe with no readers
sigquit: *INTEGER*
-- Interactive termination signal
signal_quit: *INTEGER*
-- Interactive termination signal
sigcont: *INTEGER*
-- Continue if stopped
signal_continue: *INTEGER*
-- Continue if stopped
sigstop: *INTEGER*
-- Stop signal, cannot be caught or ignored
signal_stop: *INTEGER*
-- Stop signal, cannot be caught or ignored
sigtstp: *INTEGER*
-- Interactive stop signal
signal_interactive_stop: *INTEGER*
-- Interactive stop signal
sigttin: *INTEGER*
-- Read from control terminal attempted by a member of a
-- background process group
signal_terminal_in: *INTEGER*

-- Read from control terminal attempted by a member of a
-- background process group
sigttou: *INTEGER*
-- Write to control terminal attempted by a member of a
-- background process group
signal_terminal_out: *INTEGER*
-- Write to control terminal attempted by a member of a
-- background process group
feature(*s*) **from** *POSIX_CONSTANTS*
-- sigprocmask how values
sig_block: *INTEGER*
sig_unblock: *INTEGER*
sig_setmask: *INTEGER*
feature(*s*) **from** *POSIX_CONSTANTS*
-- Posix pathconf constants
pc_name_max: *INTEGER*
-- The maximum length of a filename for this directory
feature(*s*) **from** *POSIX_CONSTANTS*
-- terminal i/o local mode flags
isig: *INTEGER*
icanon: *INTEGER*
echo: *INTEGER*
-- If set, input characters are echoed back to the terminal
echoe: *INTEGER*
echok: *INTEGER*
echonl: *INTEGER*
noflsh: *INTEGER*
tostop: *INTEGER*
iexten: *INTEGER*
feature(*s*) **from** *POSIX_CONSTANTS*
-- set terminal settings options
tcsanow: *INTEGER*
tcsadrain: *INTEGER*
tcsaflush: *INTEGER*
feature(*s*) **from** *POSIX_CONSTANTS*
-- Semaphore constants
sem_value_max: *INTEGER*
-- Valid maximum initial value for a semaphore.
feature(*s*) **from** *POSIX_CONSTANTS*
-- terminal baud rates
b0: *INTEGER*
b50: *INTEGER*
b75: *INTEGER*
b110: *INTEGER*
b134: *INTEGER*
b150: *INTEGER*
b200: *INTEGER*

b300: *INTEGER*
b600: *INTEGER*
b1200: *INTEGER*
b1800: *INTEGER*
b2400: *INTEGER*
b4800: *INTEGER*
b9600: *INTEGER*
b19200: *INTEGER*
b38400: *INTEGER*
b57600: *INTEGER*
b115200: *INTEGER*
b230400: *INTEGER*
feature(*s*) **from** *POSIX_CONSTANTS*
-- terminal i/o control mode constants
csize: *INTEGER*
cs5: *INTEGER*
cs6: *INTEGER*
cs7: *INTEGER*
cs8: *INTEGER*
cstopb: *INTEGER*
cread: *INTEGER*
parenb: *INTEGER*
parodd: *INTEGER*
hupcl: *INTEGER*
clocal: *INTEGER*
feature(*s*) **from** *POSIX_CONSTANTS*
-- terminal i/o input control flags
ignbrk: *INTEGER*
brkint: *INTEGER*
ignpar: *INTEGER*
parmrk: *INTEGER*
inpck: *INTEGER*
istrip: *INTEGER*
inlcr: *INTEGER*
igncr: *INTEGER*
icrnl: *INTEGER*
ixon: *INTEGER*
ixoff: *INTEGER*
feature(*s*) **from** *POSIX_CONSTANTS*
-- category constants
lc_messages: *INTEGER*
feature(*s*) **from** *POSIX_CONSTANTS*
-- pathname variable values
max_input: *INTEGER*
-- Minimum number of bytes for which space will be available
-- in a terminal input queue; therefore, the maximum number
-- of bytes a portable application may required to be typed

```
        -- as input before eading them
    name_max: INTEGER
        -- Maximum number of bytes in a file name
    path_max: INTEGER
        -- Maximum number of bytes in a pathname
    pipe_buf: INTEGER
        -- Maximum number of bytes that can be written atomically
        -- when writing to a pipe.
feature(s) from POSIX_CONSTANTS
    -- invariant values
    ssize_max: INTEGER
        -- The maximum value that can be stored in an object of type ssize_t
end of POSIX_CONSTANTS
```

D.5 POSIX_CURRENT_PROCESS

class *interface POSIX_CURRENT_PROCESS*
feature(*s*) **from** *STDC_CURRENT_PROCESS*
-- my stdandard input/output/error
stdin: *POSIX_TEXT_FILE*
stdout: *POSIX_TEXT_FILE*
stderr: *POSIX_TEXT_FILE*
feature(*s*) **from** *ABSTRACT_CURRENT_PROCESS*
-- process properties
pid: *INTEGER*
-- The process identifier.
is_pid_valid: *BOOLEAN*
-- current process id is always valid
feature(*s*) **from** *ABSTRACT_CURRENT_PROCESS*
-- every process also has standard file descriptors which might not be compatible with stdin/stdout/stderr (Wind
fd_stdin: *POSIX_FILE_DESCRIPTOR*
fd_stdout: *POSIX_FILE_DESCRIPTOR*
fd_stderr: *POSIX_FILE_DESCRIPTOR*
feature(*s*) **from** *STDC_SECURITY_ACCESSOR*
-- The singleton, available to any because its used in preconditions
security: *STDC_SECURITY*
-- Singleton entry point for security.
feature(*s*) **from** *STDC_BASE*
-- errno
errno: *STDC_ERRNO*
feature(*s*) **from** *ABSTRACT_PROCESS*
-- Signal this process
terminate
-- attempt to gracefully terminate this process
feature(*s*) **from** *POSIX_PROCESS*
-- signal this process
kill (*a_signal_code*: *INTEGER*)
-- Send signal *signal_code* to the process
feature(*s*) **from** *POSIX_CURRENT_PROCESS*
-- POSIX locale specifics
set_native_messages
-- Select native language as the language in which messages
-- are displayed
invariant
accessing_real_singleton: *security_is_real_singleton*;
end *of POSIX_CURRENT_PROCESS*

D.6 POSIX_DAEMON

```
deferred class interface POSIX_DAEMON
feature(s) from POSIX_DAEMON
    -- Daemon specific actions
    detach
        -- detach from command-line, not very useful if you want to
        -- spawn multiple daemons, but you can always pass daemons to
        -- the fork routine yourself.
    after_fork
        -- Code thanks to W. Richard Stevens.
        -- If you are started from inetd, youre in big trouble
        -- already and getting deeper in the mud. For inetd there will
        -- be another method to call, perhaps init_inetd or so.
invariant
    accessing_real_singleton: security_is_real_singleton;
end of deferred POSIX_DAEMON
```

D.7 POSIX_DIRECTORY

```
class interface POSIX_DIRECTORY
creation
    make (a_directory_name: STRING)
feature(s) from POSIX_DIRECTORY
    max_filename_length: INTEGER
        -- maximum length of a file in this directory
invariant
    accessing_real_singleton: security_is_real_singleton;
    dirp_remains_valid: is_open implies dirp /= default_pointer;
    directory_name_not_empty: directory_name /= Void and then not directory_name.is_empty;
    my_item_not_void: my_item /= Void;
    valid_is_dot: my_is_dot = my_item.is_equal(".");
    valid_is_dotdot: my_is_dotdot = my_item.is_equal("..");
    my_status_tracks_item: my_status /= Void implies my_status.path.is_equal(full_name);
end of POSIX_DIRECTORY
```

D.8 POSIX_EXEC_PROCESS

class *interface POSIX_EXEC_PROCESS*
creation
make (*a_program*: *STRING*; *a_arguments*: *ARRAY*[*STRING*])
make_capture_input (*a_program*: *STRING*; *a_arguments*: *ARRAY*[*STRING*])
make_capture_output (*a_program*: *STRING*; *a_arguments*: *ARRAY*[*STRING*])
make_capture_io (*a_program*: *STRING*; *a_arguments*: *ARRAY*[*STRING*])
-- Why not use threedirectional i/o, because youre getting
-- yourself in great, great trouble anyway.
-- A bit of advice: call stdin.close before starting to call
-- stdout.read_string and such...
make_capture_all (*a_program*: *STRING*; *a_arguments*: *ARRAY*[*STRING*])
-- Threedirectional i/o is a great way to get yourself in trouble.
feature(*s*) **from** *STDC_CHILD_PROCESS*
-- termination info
is_terminated: *BOOLEAN*
-- Is process not running any more?
exit_code: *INTEGER*
-- Low-order 8 bits of call to _exit or exit for this process.
feature(*s*) **from** *ABSTRACT_CHILD_PROCESS*
-- Actions that parent may execute
wait_for (*suspend*: *BOOLEAN*)
-- wait for this process to terminate. If *suspend* then we
-- wait until the information about this process is available,
-- else we return immediately.
-- If suspend is False, check the running property to see
-- if this child is really terminated.
feature(*s*) **from** *STDC_CURRENT_PROCESS*
-- my stdandard input/output/error
child_stdin: *POSIX_TEXT_FILE*
child_stdout: *POSIX_TEXT_FILE*
child_stderr: *POSIX_TEXT_FILE*
feature(*s*) **from** *ABSTRACT_CURRENT_PROCESS*
-- process properties
pid: *INTEGER*
-- either the current process identifier or the childs
is_pid_valid: *BOOLEAN*
-- current process id is always valid
feature(*s*) **from** *ABSTRACT_CURRENT_PROCESS*
-- every process also has standard file descriptors which might not be compatible with stdin/stdout/stderr (Wind
child_fd_stin: *POSIX_FILE_DESCRIPTOR*
child_fd_stdout: *POSIX_FILE_DESCRIPTOR*
child_fd_sterr: *POSIX_FILE_DESCRIPTOR*
feature(*s*) **from** *STDC_SECURITY_ACCESSOR*
-- The singleton, available to any because its used in preconditions
security: *STDC_SECURITY*

-- Singleton entry point for security.
feature(*s*) **from** *STDC_BASE*
-- errno
errno: *STDC_ERRNO*
feature(*s*) **from** *ABSTRACT_PROCESS*
-- Signal this process
terminate
-- attempt to gracefully terminate this process
feature(*s*) **from** *POSIX_PROCESS*
-- signal this process
kill (*a_signal_code*: *INTEGER*)
-- Send signal *signal_code* to the process
feature(*s*) **from** *POSIX_CURRENT_PROCESS*
-- POSIX locale specifics
set_native_messages
-- Select native language as the language in which messages
-- are displayed
feature(*s*) **from** *ABSTRACT_EXEC_PROCESS*
-- Initialization
make (*a_program*: *STRING*; *a_arguments*: *ARRAY*[*STRING*])
make_capture_input (*a_program*: *STRING*; *a_arguments*: *ARRAY*[*STRING*])
make_capture_output (*a_program*: *STRING*; *a_arguments*: *ARRAY*[*STRING*])
make_capture_io (*a_program*: *STRING*; *a_arguments*: *ARRAY*[*STRING*])
-- Why not use threedirectional i/o, because youre getting
-- yourself in great, great trouble anyway.
-- A bit of advice: call stdin.close before starting to call
-- stdout.read_string and such...
make_capture_all (*a_program*: *STRING*; *a_arguments*: *ARRAY*[*STRING*])
-- Threedirectional i/o is a great way to get yourself in trouble.
feature(*s*) **from** *ABSTRACT_EXEC_PROCESS*
-- (re)set arguments
set_arguments (*a_arguments*: *ARRAY*[*STRING*])
feature(*s*) **from** *ABSTRACT_EXEC_PROCESS*
-- i/o capturing
capture_input: *BOOLEAN*
-- is input captured on execute?
capture_output: *BOOLEAN*
-- is output captured on execute?
capture_error: *BOOLEAN*
-- is error captured on execute?
set_capture_input (*on*: *BOOLEAN*)
set_capture_output (*on*: *BOOLEAN*)
set_capture_error (*on*: *BOOLEAN*)
fd_stdin: *POSIX_FILE_DESCRIPTOR*
fd_stdout: *POSIX_FILE_DESCRIPTOR*
fd_stderr: *POSIX_FILE_DESCRIPTOR*
feature(*s*) **from** *ABSTRACT_EXEC_PROCESS*

```
    -- Execute
    execute
        -- Executes program_name
        -- dont forget to wait for this process to terminate
feature(s) from ABSTRACT_EXEC_PROCESS
    -- Accessible state
    program_name: STDC_PATH
        -- program to execute
    arguments: ARRAY[STRING]
        -- arguments to pass to program
feature(s) from POSIX_FORK_ROOT
    -- process properties
    is_valid_child_process: BOOLEAN
        -- returns True if this object seems to refer to a valid child process
        -- the real child process might have stopped though
feature(s) from POSIX_FORK_ROOT
    -- deferred routines
    after_fork
        -- chance for code to do something before the main execute
        -- mainly here for POSIX_DAEMON.
feature(s) from POSIX_FORK_ROOT
    -- termination info
    is_terminated_normally: BOOLEAN
        -- Has this process been terminated normally?
    is_exited: BOOLEAN
        -- Has this process been terminated normally?
    is_signalled: BOOLEAN
        -- Child process was terminated due to receipt of a signal
        -- that was not caught.
    signal_code: INTEGER
        -- Signal of process terminated abnormally or was stopped.
invariant
    accessing_real_singleton: security_is_real_singleton;
    program_name_not_empty: program_name /= Void and then not program_name.is_empty;
    arguments_not_void: arguments /= Void;
end of POSIX_EXEC_PROCESS
```

D.9 POSIX_FILE

```
deferred class interface POSIX_FILE
feature(s) from POSIX_FILE
    -- special makes
    make_from_file_descriptor (a_file_descriptor: ABSTRACT_FILE_DESCRIPTOR; a_mode: STRING)
        -- Open a stream from a given file descriptor.
        -- The stream will become remains leading so when the file
        -- descriptor is closed, it will not close, you have to close
        -- the strema to close the file descriptor.
invariant
    accessing_real_singleton: security_is_real_singleton;
    capacity_not_negative: capacity >= 0;
    valid_capacity: is_open = capacity > 0;
    open_implies_handle_assigned: is_open = stream /= unassigned_value;
    owned_implies_open: is_owner implies is_open;
    owned_implies_handle_assigned: is_owner implies stream /= unassigned_value;
    last_string_valid: last_string /= Void;
    gets_buf_valid: gets_buf /= Void;
end of deferred POSIX_FILE
```

D.10 POSIX_FILE_DESCRIPTOR

class *interface* *POSIX_FILE_DESCRIPTOR*
creation
open (*a_path*: *STRING*; *a_flags*: *INTEGER*)
-- open given file with access given by *flags*
open_read (*a_path*: *STRING*)
-- open given file with read-only access
open_write (*a_path*: *STRING*)
open_read_write (*a_path*: *STRING*)
open_truncate (*a_path*: *STRING*)
create_read_write (*a_path*: *STRING*)
-- Always create a file, existing or not.
-- Give read/write permissions to user only.
create_write (*a_path*: *STRING*)
-- Always create a file, existing or not.
-- Give read/write permissions to user only.
create_with_mode (*a_path*: *STRING*; *flags*, *mode*: *INTEGER*)
-- create a file according to *flags* and with *mode* access
-- permissions. Make sure you have th O_CREAT flag in flags
-- if you really want to create something!
make_as_duplicate (*another*: *ABSTRACT_FILE_DESCRIPTOR*)
-- On creation, create a duplicate from another file descriptor
-- As normal call, closes its own descriptor first (if open) and
-- duplicates next.
make_from_file (*file*: *STDC_FILE*)
-- Create file descriptor from given stream
-- The stream is leading, so this file descriptor will
-- never close itself, unless it is made an owner.
attach_to_fd (*a_fd*: *INTEGER*)
-- Create file descriptor with value *a_fd*. File descriptor
-- does not become owner, so it will not close *a_fd*.
feature(*s*) **from** *STDC_SECURITY_ACCESSOR*
-- The singleton, available to any because its used in preconditions
security: *STDC_SECURITY*
-- Singleton entry point for security.
feature(*s*) **from** *STDC_BASE*
-- errno
errno: *STDC_ERRNO*
feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*
-- Creation
open (*a_path*: *STRING*; *a_flags*: *INTEGER*)
-- open given file with access given by *flags*
open_read (*a_path*: *STRING*)
-- open given file with read-only access
open_write (*a_path*: *STRING*)
open_read_write (*a_path*: *STRING*)

open_truncate (*a_path*: *STRING*)
create_read_write (*a_path*: *STRING*)
-- Always create a file, existing or not.
-- Give read/write permissions to user only.
create_write (*a_path*: *STRING*)
-- Always create a file, existing or not.
-- Give read/write permissions to user only.
create_with_mode (*a_path*: *STRING*; *flags*, *mode*: *INTEGER*)
-- create a file according to *flags* and with *mode* access
-- permissions. Make sure you have th O_CREAT flag in flags
-- if you really want to create something!
feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*
-- Special creation
attach_to_fd (*a_fd*: *INTEGER*)
-- Create file descriptor with value *a_fd*. File descriptor
-- does not become owner, so it will not close *a_fd*.
make_as_duplicate (*another*: *ABSTRACT_FILE_DESCRIPTOR*)
-- On creation, create a duplicate from another file descriptor
-- As normal call, closes its own descriptor first (if open) and
-- duplicates next.
feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*
-- Close
close
-- We always describe an existing object, however user
-- probably wants to have control about closing a file.
detach
-- Forget the current file descriptor.
feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*
-- Change ownership of the descriptor. Can help to influence subtile garbage collector problems
make_owner
-- this file descriptor will (start to) own its descriptor
unown
-- When a stream is opened on a file descriptor the file
-- descriptor itself should not close itself, the stream
-- will close it.
feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*
-- Stream or file behaviour
is_streaming: *BOOLEAN*
-- Is data from this file descriptor coming through a network
-- stream?
set_streaming (*enable*: *BOOLEAN*)
-- Influence behaviour of certain functions if they should be
-- optimized for data coming from disk or data coming from
-- the network. In particular *is_streaming* implies that a
-- client application is prepared to handle *read*s that
-- return less than the requested number of bytes, but dont
-- assume that means end-of-file.

feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*
-- Raw read and write
last_blocked: *BOOLEAN*
-- True if last read or write call would be blocked
last_read: *INTEGER*
-- How many bytes were read by last call to *read*
-- -1 implies last_blocked
last_written: *INTEGER*
-- How many bytes were written by last call to write
-- -1 implies last_blocked
read_loop_disabled: *BOOLEAN*
-- For data coming from the network, a read does not always
-- return the requested number of bytes. In such a case,
-- reading again probably will return more bytes as they have
-- arrived in the network buffers, or perhaps by reading you
-- have freed the network buffers for more data.
-- However, certain file descriptors block when reading
-- again, because they dont return EOF when there is no more
-- data. A typical example is a character special file.
-- And sometimes the application can handle streaming reads
-- just fine and doesnt care if not all requested bytes are
-- returned.
-- This variable influences if *read* will attempt to read
-- more bytes in its loop, or not.
read (*buf*: *POINTER*; *offset*, *nbytes*: *INTEGER*)
-- Read data into *buf* at *offset* for *nbytes* bytes.
-- Number of bytes actually read are available in last_read.
-- Dont mix this routine with read_string or read_character!
write (*buf*: *POINTER*; *offset*, *nbytes*: *INTEGER*)
-- write given data from *buf* at *offset*, for *nbytes* bytes.
feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*
-- Safer read/write
read_buffer (*buf*: *STDC_BUFFER*; *offset*, *nbytes*: *INTEGER*)
-- more safe version of *read* in case you have a
-- STDC_BUFFER object
write_buffer (*buf*: *STDC_BUFFER*; *offset*, *bytes*: *INTEGER*)
-- more safe version of *write* in case you have a
-- STDC_BUFFER object
feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*
-- Read routines that can be mixed with *read* calls
last_line: *STRING*
-- last line read by *read_line (includes %N), see STRING_HELPER.chop*
read_line (*max_length*: *INTEGER*)
-- Raw, and slow, read of characters up to end of line. Can
-- be safely mixed with *read*, unlike *read_string*. Only
-- *max_length* characters are returned.
feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*

-- Eiffel like output
put (*a*: *ANY*)
-- write any Eiffel object as string
write_character (*c*: *CHARACTER*)
write_string (*s*: *STRING*)
puts (*s*: *STRING*)
put_string (*s*: *STRING*)
feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*
-- Buffered input, line reading instead of block reading, dont mix with raw *read* calls!
last_character: *CHARACTER*
last_string: *STRING*
-- last read string (includes %N), see STRING_HELPER.chop
read_character
-- Sets *last_character*.
read_string (*a_size*: *INTEGER*)
-- Implements line reading on top of *read*. Sets
-- *last_string* which includes the new line character if
-- any. Reads until eof or new line encountered, returns max
-- *a_size* characters.
feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*
-- file position
seek (*offset*: *INTEGER*)
-- set file position to given absolute *offset*
seek_from_current (*offset*: *INTEGER*)
-- set file position relative to current position
seek_from_end (*offset*: *INTEGER*)
-- set file position relative to end of file
feature(*s*) **from** *ABSTRACT_FILE_DESCRIPTOR*
-- queries
eof: *BOOLEAN*
-- True if end-of-file reached.
-- Currently Im unsure if detection is reliable
isatty: *BOOLEAN*
-- return true if handle associated with character device
is_attached_to_terminal: *BOOLEAN*
-- return true if handle associated with character device
is_blocking_io: *BOOLEAN*
-- True if blocking i/o enabled (default)
is_closed: *BOOLEAN*
-- Is file descriptor is closed?
is_open: *BOOLEAN*
-- Does *value* describes a valid file descriptor?
is_owner: *BOOLEAN*
-- Does this file descriptor own its descriptor? Only when
-- it owns the descriptor it will close it when *close* or
-- *dispose* is called
status: *POSIX_STATUS*

```
        -- The status for this file descriptor. Cached value,
        -- refreshed only when file reopened.
    value: INTEGER
        -- return the value of the file descriptor
feature(s) from ABSTRACT_FILE_DESCRIPTOR
    -- Access
    fd: INTEGER
        -- The actual file descriptor.
feature(s) from POSIX_FILE_DESCRIPTOR
    -- Initialization
    make_from_file (file: STDC_FILE)
        -- Create file descriptor from given stream
        -- The stream is leading, so this file descriptor will
        -- never close itself, unless it is made an owner.
feature(s) from POSIX_FILE_DESCRIPTOR
    -- Close
    close_on_execute
        -- close this descriptor when forking
feature(s) from POSIX_FILE_DESCRIPTOR
    -- Synchronisation
    supports_file_synchronization: BOOLEAN
        -- Do we support synchronization?
    supports_data_synchronization: BOOLEAN
        -- Do we support synchronization of data without metadata?
    synchronize
        -- Synchronize the state of a file (includes synchronize_data).
    synchronize_data
        -- Synchronize the data of a file. Cheaper than
        -- synchronize, but not always supported.
feature(s) from POSIX_FILE_DESCRIPTOR
    -- Locking
    get_lock (lock_to_test: POSIX_LOCK): POSIX_LOCK
        -- Gets lock information, returns True if a lock is set on
        -- the region in a_lock. a_lock is overwritten with that lock.
    set_lock_failed: BOOLEAN
        -- Did set_lock obtain a lock?
    attempt_lock (a_lock: POSIX_LOCK)
        -- Attempt to set lock, if not possible, set
        -- set_lock_failed.
    set_lock (a_lock: POSIX_LOCK)
        -- Attempt to set lock, wait if necessary.
feature(s) from POSIX_FILE_DESCRIPTOR
    -- non-blocking i/o
    set_blocking_io (enable: BOOLEAN)
feature(s) from POSIX_FILE_DESCRIPTOR
    -- Queries
    terminal: POSIX_TERMIOS
```

```
        -- terminal settings
    ttyname: STRING
        -- Terminal path name, or empty if this file descriptor does
        -- not refer to a terminal
invariant
    accessing_real_singleton: security_is_real_singleton;
    valid_internal_file_descriptor: fd >= -1;
    valid_open: is_open implies fd >= 0;
    valid_close: not is_open implies fd = -1;
    valid_status: is_closed implies my_status = Void;
    open_and_close_in_balance: is_open = not is_closed;
    owner_implies_open: is_owner implies is_open;
end of POSIX_FILE_DESCRIPTOR
```

D.11 POSIX_FILE_SYSTEM

class *interface POSIX_FILE_SYSTEM*
feature(*s*) **from** *STDC_SECURITY_ACCESSOR*
-- The singleton, available to any because its used in preconditions
security: *STDC_SECURITY*
-- Singleton entry point for security.
feature(*s*) **from** *STDC_BASE*
-- errno
errno: *STDC_ERRNO*
feature(*s*) **from** *STDC_FILE_SYSTEM*
-- path names
expand_path (*a_path*: *STRING*): *STDC_PATH*
-- returns a new path
feature(*s*) **from** *STDC_FILE_SYSTEM*
-- rename files/directories, remove files/directories
remove_file (*a_path*: *STRING*)
-- calls *unlink* when *a_path* is a file, or *rmdir* when
-- *a_path* is a directory.
-- error when file could not be removed (and it exists)
rename_to (*current_path*, *new_path*: *STRING*)
-- Rename a file or a directory.
-- *new_path* should not be an existing path.
feature(*s*) **from** *STDC_FILE_SYSTEM*
-- accessibility of files
is_modifiable (*a_path*: *STRING*): *BOOLEAN*
-- tests if file is readable and writable by this program
-- uses real user ID and real group ID instead of effective ones
is_readable (*a_path*: *STRING*): *BOOLEAN*
-- tests if file is readable by this program
-- uses real user ID and real group ID instead of effective ones
feature(*s*) **from** *ABSTRACT_FILE_SYSTEM*
-- directory access
change_directory (*a_directory*: *STRING*)
-- Changes the current working directory
chdir (*a_directory*: *STRING*)
-- Changes the current working directory
current_directory: *STRING*
-- The current directory
getcwd: *STRING*
-- The current directory
pwd: *STRING*
-- The current directory
make_directory (*a_directory*: *STRING*)
-- Makes a directory, only accessible by owner
mkdir (*a_directory*: *STRING*)
-- Makes a directory, only accessible by owner

remove_directory (*a_directory*: *STRING*)
-- Removes an empty directory, does not fail if directory
-- does not exist
rmdir (*a_directory*: *STRING*)
-- Removes an empty directory, does not fail if directory
-- does not exist
force_remove_directory (*a_directory*: *STRING*)
-- Removes a directory, even when not empty.
-- I suggest you do not have hard or symbolic links in *a_directory*...
feature(*s*) **from** *ABSTRACT_FILE_SYSTEM*
-- file statistics
status (*a_path*: *STRING*): *POSIX_STATUS_PATH*
-- Gets information about a file
feature(*s*) **from** *ABSTRACT_FILE_SYSTEM*
-- directory browsing
browse_directory (*a_path*: *STRING*): *POSIX_DIRECTORY*
-- Gets information about a directory
feature(*s*) **from** *ABSTRACT_FILE_SYSTEM*
-- accessibility of files
last_access_result: *INTEGER*
-- value of last access test
is_accessible (*a_path*: *STRING*; *a_mode*: *INTEGER*): *BOOLEAN*
-- Tests for file accessibility
access (*a_path*: *STRING*; *a_mode*: *INTEGER*): *BOOLEAN*
-- Tests for file accessibility
is_directory (*a_path*: *STRING*): *BOOLEAN*
-- return True if *a_path* exists and if it is a directory
is_existing (*a_path*: *STRING*): *BOOLEAN*
-- tests if file does exist, not if it is readable or writable by
-- this program!
-- uses real user ID and real group ID instead of effective ones
is_empty (*a_path*: *STRING*): *BOOLEAN*
-- True if file exists and has a size equal to zero.
is_executable (*a_path*: *STRING*): *BOOLEAN*
-- tests if file is executable by this program
is_writable (*a_path*: *STRING*): *BOOLEAN*
-- tests if file is writable by this program
-- uses real user ID and real group ID instead of effective ones
feature(*s*) **from** *ABSTRACT_FILE_SYSTEM*
-- various
is_case_sensitive: *BOOLEAN*
-- is file system case sensitive or not?
path_separator: *CHARACTER*
-- What is the path separator?
feature(*s*) **from** *ABSTRACT_FILE_SYSTEM*
-- file system properties
temporary_directory: *STRING*

```
        -- the temporary directory
feature(s) from POSIX_FILE_SYSTEM
    -- read/write permissions
    chmod (a_path: STRING; a_mode: INTEGER)
        -- Changes file mode
    change_mode (a_path: STRING; a_mode: INTEGER)
        -- Changes file mode
    permissions (a_path: STRING): POSIX_PERMISSIONS
        -- return the permissions object (a new one every time!) for
        -- the given file
    set_read_only (a_path: STRING)
        -- Make given file read_only
    set_writable (a_path: STRING)
        -- Make given file read_only
feature(s) from POSIX_FILE_SYSTEM
    -- file times
    touch (a_path: STRING)
        -- Sets the modification and access times of a_path to the
        -- current time of day.
        -- File is created if it does not exist.
    utime (a_path: STRING; access_time, modification_time: POSIX_TIME)
        -- Sets file access and modification times
feature(s) from POSIX_FILE_SYSTEM
    -- further directory access
    link (existing, new: STRING)
        -- Creates a hard link to a file
    unlink (a_path: STRING)
        -- Removes a directory entry, should be a file, not a directory.
        -- its not an error if path does not exist, but all other
        -- errors are reported
feature(s) from POSIX_FILE_SYSTEM
    -- mkfifo
    create_fifo (a_path: STRING; a_mode: INTEGER)
        -- Creates a FIFO special file.
    mkfifo (a_path: STRING; a_mode: INTEGER)
        -- Creates a FIFO special file.
feature(s) from POSIX_FILE_SYSTEM
    -- Shared memory
    unlink_shared_memory_object (name: STRING)
        -- Remove a shared memory object.
invariant
    accessing_real_singleton: security_is_real_singleton;
end of POSIX_FILE_SYSTEM
```

D.12 POSIX_FORK_ROOT

deferred class *interface POSIX_FORK_ROOT*
feature(*s*) **from** *STDC_CHILD_PROCESS*
-- termination info
is_terminated: *BOOLEAN*
-- Is process not running any more?
exit_code: *INTEGER*
-- Low-order 8 bits of call to _exit or exit for this process.
feature(*s*) **from** *ABSTRACT_CHILD_PROCESS*
-- Actions that parent may execute
wait_for (*suspend*: *BOOLEAN*)
-- wait for this process to terminate. If *suspend* then we
-- wait until the information about this process is available,
-- else we return immediately.
-- If suspend is False, check the running property to see
-- if this child is really terminated.
feature(*s*) **from** *STDC_CURRENT_PROCESS*
-- my stdandard input/output/error
stdin: *POSIX_TEXT_FILE*
stdout: *POSIX_TEXT_FILE*
stderr: *POSIX_TEXT_FILE*
feature(*s*) **from** *ABSTRACT_CURRENT_PROCESS*
-- process properties
pid: *INTEGER*
-- either the current process identifier or the childs
is_pid_valid: *BOOLEAN*
-- current process id is always valid
feature(*s*) **from** *ABSTRACT_CURRENT_PROCESS*
-- every process also has standard file descriptors which might not be compatible with stdin/stdout/stderr (Wind
fd_stdin: *POSIX_FILE_DESCRIPTOR*
fd_stdout: *POSIX_FILE_DESCRIPTOR*
fd_stderr: *POSIX_FILE_DESCRIPTOR*
feature(*s*) **from** *STDC_SECURITY_ACCESSOR*
-- The singleton, available to any because its used in preconditions
security: *STDC_SECURITY*
-- Singleton entry point for security.
feature(*s*) **from** *STDC_BASE*
-- errno
errno: *STDC_ERRNO*
feature(*s*) **from** *ABSTRACT_PROCESS*
-- Signal this process
terminate
-- attempt to gracefully terminate this process
feature(*s*) **from** *POSIX_PROCESS*
-- signal this process
kill (*a_signal_code*: *INTEGER*)

```
        -- Send signal signal_code to the process
feature(s) from POSIX_CURRENT_PROCESS
    -- POSIX locale specifics
    set_native_messages
        -- Select native language as the language in which messages
        -- are displayed
feature(s) from POSIX_FORK_ROOT
    -- process properties
    is_valid_child_process: BOOLEAN
        -- returns True if this object seems to refer to a valid child process
        -- the real child process might have stopped though
feature(s) from POSIX_FORK_ROOT
    -- deferred routines
    after_fork
        -- chance for code to do something before the main execute
        -- mainly here for POSIX_DAEMON.
    execute
        -- Start if child process.
feature(s) from POSIX_FORK_ROOT
    -- termination info
    is_terminated_normally: BOOLEAN
        -- Has this process been terminated normally?
    is_exited: BOOLEAN
        -- Has this process been terminated normally?
    is_signalled: BOOLEAN
        -- Child process was terminated due to receipt of a signal
        -- that was not caught.
    signal_code: INTEGER
        -- Signal of process terminated abnormally or was stopped.
invariant
    accessing_real_singleton: security_is_real_singleton;
end of deferred POSIX_FORK_ROOT
```

D.13 POSIX_GROUP

```
class interface POSIX_GROUP
creation
    make_from_name (a_name: STRING)
    make_from_gid (a_gid: INTEGER)
feature(s) from POSIX_GROUP
    -- creation
    make_from_name (a_name: STRING)
    make_from_gid (a_gid: INTEGER)
feature(s) from POSIX_GROUP
    -- refresh cache
    refresh
        -- refresh cache with latest info from user database
feature(s) from POSIX_GROUP
    -- queries
    name: STRING
        -- group name
    gid: INTEGER
        -- ID number
invariant
    accessing_real_singleton: security_is_real_singleton;
    valid_group: group /= default_pointer;
end of POSIX_GROUP
```

D.14 POSIX_LOCK

```
class interface POSIX_LOCK
creation
    make
feature(s) from POSIX_LOCK
    -- creation
    make
feature(s) from POSIX_LOCK
    -- members
    allow_read: BOOLEAN
        -- This is a read lock
    allow_all: BOOLEAN
        -- No lock or used to remove a lock
    allow_none: BOOLEAN
        -- This is a write lock
    start: INTEGER
    length: INTEGER
    pid: INTEGER
feature(s) from POSIX_LOCK
    -- settable members
    set_allow_read
        -- this is a read or shared lock
    set_allow_all
        -- to remove a lock
    set_allow_none
        -- this is a write or exclusive lock
    set_seek_start
        -- start is measured from the beginning of the file
    set_seek_current
        -- start is measured from the current position
    set_seek_end
        -- start is measured from the end of the file
    set_start (a_start: INTEGER)
        -- set relative offset in bytes
    set_length (a_length: INTEGER)
        -- number of bytes to lock
invariant
    accessing_real_singleton: security_is_real_singleton;
    valid_buf: buf /= Void;
    lock_type_known: allow_all or else allow_none or else allow_read;
end of POSIX_LOCK
```

D.15 POSIX_MEMORY_MAP

class *interface POSIX_MEMORY_MAP*
creation
make (*a_fd*: *POSIX_FILE_DESCRIPTOR*; *a_offset, a_size*: *INTEGER*; *a_base*: *POINTER*; *a_prot, a_flags*: *INTE*
-- Raw interface to mmap.
-- This function can fail on certain system (Linux for
-- example) if a_offset is not a multiple of PAGE_SIZE.
make_private (*a_fd*: *POSIX_FILE_DESCRIPTOR*; *a_offset, a_size*: *INTEGER*)
-- A mapping where changes are private.
-- *a_offset* denotes the offset from *a_fd*.
-- This function can fail on certain system (Linux for
-- example) if a_offset is not a multiple of PAGE_SIZE.
make_shared (*a_fd*: *POSIX_FILE_DESCRIPTOR*; *a_offset, a_size*: *INTEGER*)
-- Make a mapping where changes are shared, i.e. the
-- *a_offset* denotes the offset from *a_fd*.
-- underlying object is also changed.
-- This function can fail on certain system (Linux for
-- example) if a_offset is not a multiple of PAGE_SIZE.
feature(*s*) **from** *POSIX_MEMORY_MAP*
-- Initialization
make (*a_fd*: *POSIX_FILE_DESCRIPTOR*; *a_offset, a_size*: *INTEGER*; *a_base*: *POINTER*; *a_prot, a_flags*: *INTE*
-- Raw interface to mmap.
-- This function can fail on certain system (Linux for
-- example) if a_offset is not a multiple of PAGE_SIZE.
make_private (*a_fd*: *POSIX_FILE_DESCRIPTOR*; *a_offset, a_size*: *INTEGER*)
-- A mapping where changes are private.
-- *a_offset* denotes the offset from *a_fd*.
-- This function can fail on certain system (Linux for
-- example) if a_offset is not a multiple of PAGE_SIZE.
make_shared (*a_fd*: *POSIX_FILE_DESCRIPTOR*; *a_offset, a_size*: *INTEGER*)
-- Make a mapping where changes are shared, i.e. the
-- *a_offset* denotes the offset from *a_fd*.
-- underlying object is also changed.
-- This function can fail on certain system (Linux for
-- example) if a_offset is not a multiple of PAGE_SIZE.
feature(*s*) **from** *POSIX_MEMORY_MAP*
-- Cleanup
dispose
-- Close handle if owner.
feature(*s*) **from** *POSIX_MEMORY_MAP*
-- Unmap
close
-- Remove the mapping.
feature(*s*) **from** *POSIX_MEMORY_MAP*
-- State
offset: *INTEGER*

```
        -- Offset from file.
    fd: POSIX_FILE_DESCRIPTOR
        -- The file that is mapped.
invariant
    accessing_real_singleton: security_is_real_singleton;
    capacity_not_negative: capacity >= 0;
    valid_capacity: is_allocated = capacity > 0;
    open_implies_handle_assigned: is_allocated = ptr /= unassigned_value;
    owned_implies_open: is_owner implies is_allocated;
    owned_implies_handle_assigned: is_owner implies ptr /= unassigned_value;
    size_positive: is_open implies capacity > 0;
    ptr_valid: is_open implies ptr /= default_pointer and not is_open implies ptr = default_pointer;
    offset_not_negative: offset >= 0;
end of POSIX_MEMORY_MAP
```

D.16 POSIX_PERMISSIONS

deferred class *interface POSIX_PERMISSIONS*
feature(*s*) **from** *POSIX_PERMISSIONS*
apply
-- make permissions changes (if any) permanent
refresh
-- synchronize with permission changes possibly made on disk
feature(*s*) **from** *POSIX_PERMISSIONS*
-- query mode
allow_anyone_execute: *BOOLEAN*
-- anyone allowed to execute the file?
allow_anyone_read: *BOOLEAN*
-- anyone allowed to read the file?
allow_anyone_read_write: *BOOLEAN*
-- anyone allowed to read and write the file?
allow_anyone_write: *BOOLEAN*
-- anyone allowed to write the file?
allow_group_execute: *BOOLEAN*
-- process with a group ID that matches the files group
-- allowed to execute the file?
allow_group_read: *BOOLEAN*
-- process with a group ID that matches the files group
-- allowed to read the file?
allow_group_read_write: *BOOLEAN*
-- process with a group ID that matches the files group
-- allowed to read the file?
allow_group_write: *BOOLEAN*
-- process with a group ID that matches the files group
-- allowed to write the file?
allow_owner_execute: *BOOLEAN*
-- owner allowed to execute the file
allow_read: *BOOLEAN*
allow_owner_read: *BOOLEAN*
allow_read_write: *BOOLEAN*
allow_owner_read_write: *BOOLEAN*
allow_write: *BOOLEAN*
allow_owner_write: *BOOLEAN*
is_set_group_id: *BOOLEAN*
-- group ID set on execution?
is_set_gid: *BOOLEAN*
-- group ID set on execution?
is_set_user_id: *BOOLEAN*
-- user ID set on execution?
is_set_uid: *BOOLEAN*
-- user ID set on execution?
feature(*s*) **from** *POSIX_PERMISSIONS*

-- set permissions
set_allow_anyone_execute (*allow*: *BOOLEAN*)
-- give anyone execute permission
set_allow_anyone_read (*allow*: *BOOLEAN*)
-- give anyone read permission
set_allow_anyone_read_write (*allow*: *BOOLEAN*)
-- give anyone read and write permissions
set_allow_anyone_write (*allow*: *BOOLEAN*)
-- give anyone write permission
set_allow_group_execute (*allow*: *BOOLEAN*)
-- give group execute permission
set_allow_group_read (*allow*: *BOOLEAN*)
-- give group read permission
set_allow_group_read_write (*allow*: *BOOLEAN*)
-- give group read and write permission
set_allow_group_write (*allow*: *BOOLEAN*)
-- give group write permission
set_allow_owner_execute (*allow*: *BOOLEAN*)
-- give owner execute permission
set_allow_read (*allow*: *BOOLEAN*)
-- give read permission
set_allow_owner_read (*allow*: *BOOLEAN*)
-- give read permission
set_allow_read_write (*allow*: *BOOLEAN*)
-- give read/write permission
set_allow_write (*allow*: *BOOLEAN*)
-- give write permission
set_allow_owner_write (*allow*: *BOOLEAN*)
-- give write permission
feature(*s*) **from** *POSIX_PERMISSIONS*
-- direct access to Unix fields
uid: *INTEGER*
-- id of object owner, always 0 on NT
owner_id: *INTEGER*
-- id of object owner, always 0 on NT
gid: *INTEGER*
-- id of group, always 0 on NT
group_id: *INTEGER*
-- id of group, always 0 on NT
mode: *INTEGER*
-- the bit coded Unix mode field
feature(*s*) **from** *POSIX_PERMISSIONS*
-- set owner and group
set_owner_id (*a_owner_id*: *INTEGER*)
-- change the owner
set_group_id (*a_group_id*: *INTEGER*)
-- change the group

invariant
accessing_real_singleton: *security_is_real_singleton*;
end *of* **deferred** *POSIX_PERMISSIONS*

D.17 POSIX_PIPE

```
class interface POSIX_PIPE
creation
    make
        -- Create pipe
feature(s) from POSIX_PIPE
    -- the pipe
    fdin: POSIX_FILE_DESCRIPTOR
    fdout: POSIX_FILE_DESCRIPTOR
invariant
    accessing_real_singleton: security_is_real_singleton;
    valid_pipe: fdin /= Void and fdout /= Void;
end of POSIX_PIPE
```

D.18 POSIX_SEMAPHORE

```
class interface POSIX_SEMAPHORE
feature(s) from POSIX_SEMAPHORE
    -- commands
    attempt_acquire
        -- Lock the semaphore only if it is not locked. If it is locked
        -- by some process, this command returns immediately and the
        -- semaphore is not locked
    acquire
        -- lock the semaphore
    release
        -- unlock the semaphore
feature(s) from POSIX_SEMAPHORE
    -- queries
    is_initialized: BOOLEAN
        -- True if semaphore is initialized/opened/created
    is_locked: BOOLEAN
        -- True if this process has locked the semaphore
    supports_semaphores: BOOLEAN
        -- True if semaphores are supported
        -- most systems support unnamed semaphores, but still return False here
    value: INTEGER
        -- value of semaphore if not locked.
        -- Value is <= 0 if this semaphore is locked.
invariant
    accessing_real_singleton: security_is_real_singleton;
    sem_value_valid: sem_value /= Void;
end of POSIX_SEMAPHORE
```

D.19 POSIX_SIGNAL

class *interface POSIX_SIGNAL*
creation
make (*a_value*: *INTEGER*)
feature(*s*) **from** *POSIX_SIGNAL*
-- Initialization
make (*a_value*: *INTEGER*)
feature(*s*) **from** *POSIX_SIGNAL*
-- Set signal properties, make effective with *apply*
apply
-- Make changes effective.
set_child_stop (*stop*: *BOOLEAN*)
-- Generate SIGCHLD when children stop.
set_default_action
-- Install signal-specific default action when *apply* is called.
set_ignore_action
-- Ignore signal when *apply* is called..
set_handler (*a_handler*: *STDC_SIGNAL_HANDLER*)
-- Install ones own signal handler when *apply* is called.
set_mask (*a_mask*: *POSIX_SIGNAL_SET*)
feature(*s*) **from** *POSIX_SIGNAL*
-- signal functions
raise_in (*a_pid*: *INTEGER*)
-- Raise the signal in the given process.
feature(*s*) **from** *POSIX_SIGNAL*
-- Signal state
child_stop: *BOOLEAN*
-- generate SIGCHLD when children stop
handler: *POINTER*
-- pointer to function which catches this signal
is_defaulted: *BOOLEAN*
-- signal is handled by its specific default action
is_ignored: *BOOLEAN*
-- signal is ignored
is_ignorable: *BOOLEAN*
-- True if this signal is ignorable, either it is so by
-- default or it may be set so.
mask: *POSIX_SIGNAL_SET*
refresh
-- get latest state for this signal
invariant
accessing_real_singleton: *security_is_real_singleton*;
accessing_real_singleton: *signal_switch_is_real_singleton*;
valid_signal_value: *value >= 1*;
has_memory: *sigaction /= Void*;
end *of POSIX_SIGNAL*

D.20 POSIX_SIGNAL_SET

class *interface POSIX_SIGNAL_SET*
creation
make_empty
-- make an initially empty signal set
make_full
-- make a set where all signals are enabled
make_pending
-- this signal set will be the set of signals that are blocked
-- and pending
feature(*s*) **from** *POSIX_SIGNAL_SET*
-- creation, make a set
make_empty
-- make an initially empty signal set
make_full
-- make a set where all signals are enabled
make_pending
-- this signal set will be the set of signals that are blocked
-- and pending
feature(*s*) **from** *POSIX_SIGNAL_SET*
-- change a set
extend (*signo*: *INTEGER*)
-- add signal to set
put (*signo*: *INTEGER*)
-- add signal to set
prune (*signo*: *INTEGER*)
-- remove the signal from the set
wipe_out
-- remove all items
feature(*s*) **from** *POSIX_SIGNAL_SET*
-- commands to do something with set
add_to_blocked_signals
-- Add the signals to the set of blocked signals
remove_from_blocked_signals
-- Remove the signals from the set of blocked signals
set_blocked_signals
-- Set the set of blocked signals to this set
suspend
-- Suspend process, until delivery of a signal whose action
-- is either to execute a signal-catching function or to
-- terminate the process
feature(*s*) **from** *POSIX_SIGNAL_SET*
-- queries
has (*signo*: *INTEGER*): *BOOLEAN*
-- is signal *signo* in the set
invariant

```
    accessing_real_singleton: security_is_real_singleton;
    have_set: set /= Void;
end of POSIX_SIGNAL_SET
```

D.21 POSIX_STATUS

```
deferred class interface POSIX_STATUS
feature(s) from POSIX_STATUS
    -- stat members
    is_block_special: BOOLEAN
        -- True if block-special file
    ino: INTEGER
    inode: INTEGER
    permissions: POSIX_PERMISSIONS
        -- file permissions
        ensure
            valid_result: Result /= Void
feature(s) from POSIX_STATUS
    -- direct access to the unix fields, not recommended
    unix_gid: INTEGER
    unix_uid: INTEGER
invariant
    accessing_real_singleton: security_is_real_singleton;
    valid_stat: stat /= Void and then stat.capacity >= abstract_stat_size;
end of deferred POSIX_STATUS
```

D.22 POSIX_SYSTEM

class *interface POSIX_SYSTEM*
feature(*s*) **from** *POSIX_SYSTEM*
-- Sysconf queries, run-time determined
child_max: *INTEGER*
-- The number of simultaneous processes per real user ID.
clock_ticks: *INTEGER*
-- The number of clock ticks per second.
has_job_control: *BOOLEAN*
-- Job control functions are supported.
has_saved_ids: *BOOLEAN*
-- Each process has a saved set-user-ID and a saved set-group-ID.
ngroups_max: *INTEGER*
-- The number of simultaneous supplementary group IDs.
page_size: *INTEGER*
-- granularity in bytes of memory mapping and process memory locking.
posix_version: *INTEGER*
-- Indicates the 4-digit year and 2-digit month that the
-- standard was approved.
feature(*s*) **from** *POSIX_SYSTEM*
-- Compile-time determined queries
supports_asynchronous_io: *BOOLEAN*
-- True if the message passing API is supported.
supports_file_synchronization: *BOOLEAN*
-- True if file synchronization is supported.
supports_memory_mapped_files: *BOOLEAN*
-- True if memory mapped files are supported.
supports_memory_locking: *BOOLEAN*
-- True if memory locking is supported.
supports_memlock_range: *BOOLEAN*
-- True if memory range locking is supported.
supports_memory_protection: *BOOLEAN*
-- True if memory protection is supported.
supports_message_passing: *BOOLEAN*
-- True if the message passing API is supported.
supports_priority_scheduling: *BOOLEAN*
-- True if priority scheduling is supported.
supports_semaphores: *BOOLEAN*
-- True if semaphores are supported.
supports_shared_memory_objects: *BOOLEAN*
-- True if shared memory objects are supported.
supports_synchronized_io: *BOOLEAN*
-- True if synchronized io is supported.
supports_timers: *BOOLEAN*
-- True if timers are supported.
supports_threads: *BOOLEAN*

```
        -- True if thread are supported.
invariant
    accessing_real_singleton: security_is_real_singleton;
end of POSIX_SYSTEM
```

D.23 POSIX_TERMIOS

class *interface* *POSIX_TERMIOS*
creation
make (*a_fd*: *POSIX_FILE_DESCRIPTOR*)
feature(*s*) **from** *POSIX_TERMIOS*
-- creation
make (*a_fd*: *POSIX_FILE_DESCRIPTOR*)
feature(*s*) **from** *POSIX_TERMIOS*
-- raw individual fields
iflag: *INTEGER*
-- input mode flags
oflag: *INTEGER*
-- output mode flags
cflag: *INTEGER*
-- control mode flags
lflag: *INTEGER*
-- local mode flags
feature(*s*) **from** *POSIX_TERMIOS*
-- more friendly settings
is_input_echoed: *BOOLEAN*
-- are input characters echoed back to the terminal?
is_receiving: *BOOLEAN*
-- If false, no characters are received
set_echo_input (*enable*: *BOOLEAN*)
set_echo_new_line (*enable*: *BOOLEAN*)
set_input_control (*enable*: *BOOLEAN*)
-- enable start/stop input control
set_receive (*enable*: *BOOLEAN*)
feature(*s*) **from** *POSIX_TERMIOS*
-- line control functions
flush_input
-- discards all data that has been received but not read
drain
-- wait for all output to be transmitted to the terminal
send_break
-- sends a break to the terminal
feature(*s*) **from** *POSIX_TERMIOS*
-- get/set baudrates as symbols
input_speed: *INTEGER*
-- returns terminal input baud rate as symbolic value
output_speed: *INTEGER*
-- returns terminal output baud rate as symbolic value
set_input_speed (*new_rate*: *INTEGER*)
-- sets terminal input baud rate, *new_rate* is one of the
-- BXXXX constants
set_output_speed (*new_rate*: *INTEGER*)

```
        -- sets terminal output baud rate, new_rate is one of the
        -- BXXXX constants
feature(s) from POSIX_TERMIOS
    -- symbol to baud rate conversions
    speed_to_baud_rate (symbol: INTEGER): INTEGER
        -- given a baud rate symbol, the real baud rate is returned.
feature(s) from POSIX_TERMIOS
    -- apply/refresh state
    apply_now
        -- change occurs immediately
    apply_drain
        -- change occurs after all output written to fd has been
        -- transmitted. This function should be used when changing
        -- parameters that affect output.
    apply_flush
        -- change occurs after all output written to fd has been
        -- transmitted. All input that has been received but not
        -- read, is discarded before the change is made.
    refresh
        -- get terminal settings currently in effect
feature(s) from POSIX_TERMIOS
    -- state
    fd: POSIX_FILE_DESCRIPTOR
        -- the file descriptor for these terminal settings
invariant
    accessing_real_singleton: security_is_real_singleton;
    valid_attr: attr /= Void and then attr.capacity = posix_termios_size;
    valid_fd: fd /= Void;
end of POSIX_TERMIOS
```

D.24 POSIX_TIMED_COMMAND

```
deferred class interface POSIX_TIMED_COMMAND
feature(s) from POSIX_TIMED_COMMAND
    -- Initialization
    make (a_seconds: INTEGER)
feature(s) from POSIX_TIMED_COMMAND
    -- Execution
    execute: BOOLEAN
        -- Did do_execute complete its task within seconds seconds?
feature(s) from POSIX_TIMED_COMMAND
    -- Access
    is_signal_alarm_handled: BOOLEAN
        -- Does the signal SIGNAL_ALARM cause an Eiffel exception?
feature(s) from POSIX_TIMED_COMMAND
    -- State
    remaining_seconds: INTEGER
        -- number of seconds left in previous request
    seconds: INTEGER
        -- the number of seconds available to execute the command
    set_seconds (a_seconds: INTEGER)
invariant
    accessing_real_singleton: security_is_real_singleton;
    valid_seconds: seconds >= 1;
end of deferred POSIX_TIMED_COMMAND
```

D.25 POSIX_USER

class *interface POSIX_USER*
creation
make_from_name (*a_name*: *STRING*)
make_from_uid (*a_uid*: *INTEGER*)
feature(*s*) **from** *POSIX_USER*
-- creation
make_from_name (*a_name*: *STRING*)
make_from_uid (*a_uid*: *INTEGER*)
feature(*s*) **from** *POSIX_USER*
-- refresh cache
refresh
-- refresh cache with latest info from user database
feature(*s*) **from** *POSIX_USER*
-- queries
name: *STRING*
-- login name
uid: *INTEGER*
-- ID number
gid: *INTEGER*
-- group ID number
home_directory: *STRING*
-- initial working directory
shell: *STRING*
-- initial user program
invariant
accessing_real_singleton: *security_is_real_singleton*;
valid_passwd: *passwd* /= *default_pointer*;
end *of POSIX_USER*

D.26 POSIX_USER_DATABASE

```
class interface POSIX_USER_DATABASE
feature(s) from POSIX_USER_DATABASE
    -- queries
    is_existing_uid (uid: INTEGER): BOOLEAN
        -- Returns True if this uid exists in /etc/passwd
        -- (or through NIS or whatever mechanisms that might be in use)
    is_existing_login (login: STRING): BOOLEAN
        -- Returns True if this login exists in /etc/passwd
        -- (or through NIS or whatever mechanisms that might be in use)
invariant
    accessing_real_singleton: security_is_real_singleton;
end of POSIX_USER_DATABASE
```

In this chapter:

E Short (flat) listing of Single Unix Specification classes

Classes in this appendix are based on the Single Unix Specification. They inherit from the POSIX classes. Inherited features are not shown.

E.1 SUS_CONSTANTS

class *interface SUS_CONSTANTS*
feature(*s*) **from** *SUS_CONSTANTS*
-- syslog facility codes
log_kern: *INTEGER*
-- kernel messages
log_user: *INTEGER*
-- random user-level messages
log_mail: *INTEGER*
-- mail system
log_daemon: *INTEGER*
-- system daemons
log_auth: *INTEGER*
-- security/authorization messages
log_lpr: *INTEGER*
-- line printer subsystem
log_news: *INTEGER*
-- network news subsystem
log_uucp: *INTEGER*
-- UUCP subsystem
log_cron: *INTEGER*
-- clock daemon
log_local0: *INTEGER*
-- Reserved for local use
log_local1: *INTEGER*
-- Reserved for local use
log_local2: *INTEGER*
-- Reserved for local use
log_local3: *INTEGER*

```
        -- Reserved for local use
    log_local4: INTEGER
        -- Reserved for local use
    log_local5: INTEGER
        -- Reserved for local use
    log_local6: INTEGER
        -- Reserved for local use
    log_local7: INTEGER
        -- Reserved for local use
feature(s) from SUS_CONSTANTS
    -- syslog open options
    log_pid: INTEGER
        -- log the pid with each message
    log_cons: INTEGER
        -- log on the console if errors in sending
    log_odelay: INTEGER
        -- delay open until first syslog() (default)
    log_ndelay: INTEGER
        -- dont delay open
end of SUS_CONSTANTS
```

E.2 SUS_ENV_VAR

```
class interface SUS_ENV_VAR
creation
    make (a_name: STRING)
feature(s) from SUS_ENV_VAR
    -- commands
    set_value (new_value: STRING)
invariant
    accessing_real_singleton: security_is_real_singleton;
end of SUS_ENV_VAR
```

E.3 SUS_FILE_SYSTEM

```
class interface SUS_FILE_SYSTEM
feature(s) from SUS_FILE_SYSTEM
    -- file statistics
    status (a_path: STRING): SUS_STATUS_PATH
        -- Return information about path.
    symbolic_link_status (a_path: STRING): SUS_STATUS
        -- Return information about path, but if it is a symbolic
        -- link, about the symbolic link instead of the referenced path
feature(s) from SUS_FILE_SYSTEM
    -- symbolic links
    create_symbolic_link (old_path, new_path: STRING)
        -- Creates a symbolic link
    symlink (old_path, new_path: STRING)
        -- Creates a symbolic link
feature(s) from SUS_FILE_SYSTEM
    -- path names
    resolved_path_name (a_path: STRING): STRING
        -- derives from a_path an absolute pathname that names the
        -- same file, whose resolution does not involve ".", "..", or
        -- symbolic links.
    realpath (a_path: STRING): STRING
        -- derives from a_path an absolute pathname that names the
        -- same file, whose resolution does not involve ".", "..", or
        -- symbolic links.
invariant
    accessing_real_singleton: security_is_real_singleton;
end of SUS_FILE_SYSTEM
```

E.4 SUS_HOST

```
class interface SUS_HOST
creation
    make_from_name (a_name: STRING)
    make_from_address (a_address: SUS_IP_ADDRESS)
feature(s) from SUS_HOST
    -- creation
    make_from_name (a_name: STRING)
    make_from_address (a_address: SUS_IP_ADDRESS)
feature(s) from SUS_HOST
    -- Command
    find_by_name
        -- Attempt to lookup up the host given in name. Sets
        -- found if host could be found.
        -- If found, sets canonical_name, aliases,
        -- address_type, address_length and addresses.
feature(s) from SUS_HOST
    -- Queries
    found: BOOLEAN
        -- True if name was found.
    name: STRING
        -- Name as given to make_from_name or else equal to
        -- canonical_name.
    canonical_name: STRING
        -- Official (canonical) name of host.
    aliases: ARRAY[STRING]
        -- Alias names.
    address_type: INTEGER
        -- Host address type: AF_INET or AF_INET6
    address_length: INTEGER
        -- Length of address: 4 or 16.
    addresses: ARRAY[SUS_IP_ADDRESS]
        -- Array with IPv4 or IPv6 addresses.
invariant
    accessing_real_singleton: security_is_real_singleton;
    name_not_empty: name /= Void and then not name.is_empty;
    has_canonical_name: found = canonical_name /= Void;
    has_at_least_one_ip_address: found = addresses /= Void and then addresses.count > 0;
    has_aliases: found = aliases /= Void;
    valid_length: found implies address_length > 0;
    consistent: addresses /= Void and then addresses.count > 0 implies found;
end of SUS_HOST
```

E.5 SUS_SERVICE

```
class interface SUS_SERVICE
creation
    make_from_name (a_name, a_protocol: STRING)
        -- Find service with a_name and optional a_protocol or raise
        -- exception.
    make_from_port (a_port: INTEGER; a_protocol: STRING)
        -- Initialize service from given a_port
feature(s) from SUS_SERVICE
    -- creation
    make_from_name (a_name, a_protocol: STRING)
        -- Find service with a_name and optional a_protocol or raise
        -- exception.
    make_from_port (a_port: INTEGER; a_protocol: STRING)
        -- Initialize service from given a_port
feature(s) from SUS_SERVICE
    -- public state, all data in host byte order
    port: INTEGER
        -- port number
    name: STRING
        -- official service name
    aliases: ARRAY[STRING]
        -- alias list
    protocol: STRING
        -- protocol to use (udp/tcp)
    protocol_type: INTEGER
        -- SOCK_STREAM or SOCK_DGRAM
invariant
    accessing_real_singleton: security_is_real_singleton;
    name_void_or_not_empty: name = Void or else not name.is_empty;
    valid_port: port >= 0 and port <= 65535;
    valid_protocol: protocol = Void or else protocol.is_empty or else protocol.is_equal(once_tcp) or protocol.is_equ
    valid_protocol_type: protocol_type = sock_stream or else protocol_type = sock_dgram;
    valid_aliases: aliases /= Void;
end of SUS_SERVICE
```

E.6 SUS_SOCKET_ADDRESS

class *interface SUS_SOCKET_ADDRESS*
creation
make (*a_host*: *SUS_HOST*; *a_service*: *SUS_SERVICE*)
-- Initialize socket for known host.
feature(*s*) **from** *SUS_SOCKET_ADDRESS*
-- Creation
make (*a_host*: *SUS_HOST*; *a_service*: *SUS_SERVICE*)
-- Initialize socket for known host.
feature(*s*) **from** *SUS_SOCKET_ADDRESS*
-- Public state
host: *SUS_HOST*
-- Resolved host name.
service: *SUS_SERVICE*
-- Port and protocol (udp/tcp) type.
feature(*s*) **from** *SUS_SOCKET_ADDRESS*
-- Fill socket structure, so *ptr* returns something valid
set_address (*item*: *INTEGER*)
-- Use one of the ip addresses of *host* as the socket address.
feature(*s*) **from** *SUS_SOCKET_ADDRESS*
-- Features the C API calls like
length: *INTEGER*
-- Size of my struct sockaddr_in.
ptr: *POINTER*
-- Points to struct sockaddr_in or sockaddr_in6.
invariant
accessing_real_singleton: *security_is_real_singleton*;
host_found: *host* /= *Void* **and then** *host.found*;
has_service: *service* /= *Void*;
valid_buf: *buf* /= *Void* **and then** *buf.capacity* >= *length*;
end *of SUS_SOCKET_ADDRESS*

E.7 SUS_SYSLOG

```
class interface SUS_SYSLOG
feature(s) from SUS_SYSLOG
    -- open and close
    open (a_identification: STRING; a_format, a_facility: INTEGER)
        -- start logging with the given identification
    close
        -- stop logging
feature(s) from SUS_SYSLOG
    -- Write log messages, will auto-open if not is_open
    emergency (msg: STRING)
        -- the system is unusable
    alert (msg: STRING)
        -- action must be taken immediately
    critical (msg: STRING)
        -- critical conditions
    error (msg: STRING)
        -- error conditions
    warning (msg: STRING)
        -- warning conditions
    notice (msg: STRING)
        -- normal but significant condition
    info (msg: STRING)
        -- informational
    debug_dump (msg: STRING)
        -- Debug-level messages.
feature(s) from SUS_SYSLOG
    -- state
    identification: STRING
    format: INTEGER
    facility: INTEGER
    is_open: BOOLEAN
invariant
    accessing_real_singleton: security_is_real_singleton;
    remain_single: Current = the_singleton;
    have_identification: is_open implies identification /= Void and then not identification.is_empty;
end of SUS_SYSLOG
```

E.8 SUS_TCP_SOCKET

class *interface SUS_TCP_SOCKET*
creation
listen_by_address (*sa*: *SUS_SOCKET_ADDRESS*)
-- Listen on socket for address specified in *sa*.
open_by_address (*sa*: *SUS_SOCKET_ADDRESS*)
-- Open socket to server specified in *sa*.
invariant
accessing_real_singleton: *security_is_real_singleton*;
valid_internal_file_descriptor: *fd >= -1*;
valid_open: *is_open* **implies** *fd >= 0*;
valid_close: **not** *is_open* **implies** *fd = -1*;
valid_status: *is_closed* **implies** *my_status = Void*;
open_and_close_in_balance: *is_open* = **not** *is_closed*;
owner_implies_open: *is_owner* **implies** *is_open*;
end *of SUS_TCP_SOCKET*

F
Short (flat) listing of Standard C bonus classes

In this chapter:

Classes in this appendix are based on Standard C only.

F.1 EPX_CGI

```
deferred class interface EPX_CGI
feature(s) from EPX_CGI
    -- the output routine
    execute
        -- to be implemented by child
feature(s) from EPX_CGI
    -- debug support
    dump_input
        -- Write cgi input to /tmp/cgi_input.
        -- First line contains the content header, is not actually in input!
feature(s) from EPX_CGI
    -- Standard variables
    auth_type: STRING
        -- type of authentication used
    content_type: STRING
        -- MIME type of data when invoked with POST method
    content_length: INTEGER
        -- length, in bytes, of data when invoked with POST method
    gateway_interface: STRING
        -- name and version of the gateway, for example CGI/1.1
    http_accept: STRING
        -- contents of the Accept header line sent by the client
    http_referer: STRING
        -- contents of the Referer header line
    http_user_agent: STRING
        -- name of the client program that is making the request
    path_info: STRING
        -- extra path information as it was passed to the server in
        -- the query URL
```

path_translated: *STRING*
-- extra path information translated to a final, usable
-- form. The Web document root is prepended to the query
-- path, and any other path translations are executed.
query_string: *STRING*
-- the input when invoked with the GET method
remote_addr: *STRING*
-- IP address of the client that made the request
remote_address: *STRING*
-- IP address of the client that made the request
remote_host: *STRING*
-- name of the remote computer that made the request
remote_ident: *STRING*
-- user name as given by the ident protocol
remote_user: *STRING*
-- name of the remote user that made the request
request_method: *STRING*
-- name of the method used to invoke the CGI
-- application. Valid values are GET and POST
script_name: *STRING*
-- name of script that was invoked
server_name: *STRING*
-- domain name of the computer that is running the server software
server_port: *INTEGER*
-- TCP port number on which the server that invoked the CGI
-- application is operating
server_protocol: *STRING*
-- name of the protocol that the server is using and the
-- version of that protocol. The protocol name and version
-- are separated by a forward slash with no spaces, for
-- instance HTTP/1.0
server_software: *STRING*
-- name of the server that is handling the request
feature(*s*) **from** *EPX_CGI*
-- Standard cgi header
content_text_html
content_text_plain
feature(*s*) **from** *EPX_CGI*
-- Server push, multipart header
content_multipart_x_mixed_replace (*boundary*: *STRING*)
content_next_part
-- write boundary so next part of multipart msg can be written
content_multipart_end
-- write boundary of multipart
is_multipart_message: *BOOLEAN*
feature(*s*) **from** *EPX_CGI*
-- Form input

```
    has_input: BOOLEAN
        -- True if input passed to cgi program.
    has_key (key: STRING): BOOLEAN
        -- Returns True if key defined in input.
    is_meta_char (c: CHARACTER): BOOLEAN
        -- Return True if c is a commonly used meta characters.
    meta_chars: STRING
    raw_value (key: STRING): STRING
        -- Returns value for key.
        -- if key does not exist, the empty string is returned.
    remove_meta_chars (s: STRING)
        -- If s contains meta characters, theyre removed.
    value (key: STRING): STRING
        -- Returns safe value for key, meta characters are removed.
invariant
    -- lower_a_code_definition: lower_a_code = (a).code
    -- Same thing for all other codes.
    -- (see "note" in indexing clause.)
    accessing_real_singleton: security_is_real_singleton;
    my_xml_not_void: my_xml /= Void;
    same_size: attributes.count = values.count;
    has_tag_stack: tags /= Void;
    fragment_has_no_header: is_fragment implies is_header_written;
    values_not_void: values /= Void;
    attributes_not_void: attributes /= Void;
    every_attribute_has_a_value: attributes.count = values.count;
end of deferred EPX_CGI
```

F.2 EPX_SOAP_WRITER

class *interface EPX_SOAP_WRITER*
creation
make
-- Create an XML document with initial capacity of 1024 characters.
make_with_capacity (*a_capacity*: *INTEGER*)
-- Create an XML document with initial capacity of
-- *a_capacity* characters.
feature(*s*) **from** *EPX_SOAP_WRITER*
-- SOAP specific calls
start_envelope
stop_envelope
start_header
stop_header
start_body
stop_body
feature(*s*) **from** *EPX_SOAP_WRITER*
-- SOAP header attributes
set_must_understand (*value*: *BOOLEAN*)
-- Set the SOAP-Env:mustUnderstand attribute to *value*.
feature(*s*) **from** *EPX_SOAP_WRITER*
-- Queries if tags started
is_envelope_started: *BOOLEAN*
is_header_started: *BOOLEAN*
is_body_started: *BOOLEAN*
feature(*s*) **from** *EPX_SOAP_WRITER*
-- SOAP tags
soap_env_body: *STRING*
soap_env_envelope: *STRING*
soap_env_header: *STRING*
feature(*s*) **from** *EPX_SOAP_WRITER*
-- SOAP name space
soap_env: *STRING*
soap_name_space: *STRING*
invariant
-- lower_a_code_definition: lower_a_code = (a).code
-- Same thing for all other codes.
-- (see "note" in indexing clause.)
accessing_real_singleton: *security_is_real_singleton*;
my_xml_not_void: *my_xml* /= *Void*;
same_size: *attributes.count* = *values.count*;
has_tag_stack: *tags* /= *Void*;
fragment_has_no_header: *is_fragment* **implies** *is_header_written*;
values_not_void: *values* /= *Void*;
attributes_not_void: *attributes* /= *Void*;

```
    every_attribute_has_a_value: attributes.count = values.count;
end of EPX_SOAP_WRITER
```

F.3 EPX_URI

```
class interface EPX_URI
creation
    make (a_reference: STRING)
        -- Create an absolute or relative URI.
    make_resolve (base: EPX_URI; a_reference: STRING)
        -- If a_reference is a partial URI, it is resolved using base.
feature(s) from EPX_URI
    -- Initialization.
    make (a_reference: STRING)
        -- Create an absolute or relative URI.
    make_resolve (base: EPX_URI; a_reference: STRING)
        -- If a_reference is a partial URI, it is resolved using base.
feature(s) from EPX_URI
    -- Queries
    is_absolute: BOOLEAN
        -- True if this is an absolute URI.
    is_relative: BOOLEAN
        -- True if this is a relative URI.
    has_absolute_path: BOOLEAN
        -- True if this URI has an absolute path.
feature(s) from EPX_URI
    -- Most generic URI components
    full_reference: STRING
        -- The entire thing.
    scheme: STRING
        -- Scheme used, like "http" or "ftp", anything before the :.
    scheme_specific_part: STRING
        -- Interpretation depends on scheme, everything after the :
        -- and before the ?
feature(s) from EPX_URI
    -- If URI has a hierarchical relationships within the namespace
    authority: STRING
        -- Authority part of scheme_specific_part, usually a host name.
        -- It can be more complex however like: <userinfo>@<host>:<port>.
        -- Use parse_authority to split authority in these
        -- components if that is applicable for the protocol.
    path: STRING
        -- Path in scheme_specific_part, consisting of names
        -- separated by slashes.
    query: STRING
        -- Anything after the ? if present, else Void
    fragment: STRING
        -- The part after the # if present, else Void
feature(s) from EPX_URI
    -- If authority is <userinfo>@<host>:<port>
```

```
    user_info: STRING
        -- Usually a user name.
    host: STRING
        -- hostname or IP4 address. IP6 addresses are explicitly not
        -- supported by RFC 2396
    port: INTEGER
        -- TCP port, 0 if no port present.
    is_server_authority: BOOLEAN
        -- True if authority can be parsed as:
        -- [ userinfo @ ] host [ : port ]
        -- and port, if present, is an integer.
    parse_authority (default_port: INTEGER)
        -- Assume authority can be parsed as:
        -- [ userinfo @ ] host [ : port ].
        -- If assumption is untrue, you get a nice exception...
        -- default_port is 0 means no default.
invariant
    either_absolute_or_relative: is_absolute xor is_relative;
    reference_not_empty: full_reference /= Void and then not full_reference.is_empty; -- Im really unsure if these c
 -- Constraints on elements of a parsed URI.
    valid_authority: authority = Void or else not authority.is_empty;
    valid_path: path = Void or else not path.is_empty;
    valid_query: query = Void or else not query.is_empty;
    valid_fragment: fragment = Void or else not fragment.is_empty; -- Contraints on parsed authority
    user_info_occurs_in_authority: user_info /= Void implies authority.substring_index(user_info,1) /= 0;
    host_occurs_in_authority: host /= Void implies authority.substring_index(host,1) /= 0;
    valid_port: port >= 0 and port <= 65535;
end of EPX_URI
```

F.4 EPX_XML_WRITER

class *interface EPX_XML_WRITER*
creation
make
-- Create an XML document with initial capacity of 1024 characters.
make_with_capacity (*a_capacity*: *INTEGER*)
-- Create an XML document with initial capacity of
-- *a_capacity* characters.
make_fragment
-- Create an XML fragment (document without header) with
-- initial capacity of 1024 characters.
make_fragment_with_capacity (*a_capacity*: *INTEGER*)
-- Create an XML fragment (document without header) with
-- initial capacity of *a_capacity* characters.
feature(*s*) **from** *EPX_XML_WRITER*
-- Constants from the XML specification, should be Unicode...
validfirstchars: *STRING*
-- Which characters are valid as the first character.
validotherchars: *STRING*
-- Which characters are valid as second etc characters.
feature(*s*) **from** *EPX_XML_WRITER*
-- Queries
has_invalid_control_characters (*s*: *STRING*): *BOOLEAN*
-- Contains *s* characters in the range 0x00-0x1F other then
-- TAB (0x09), CR (0x0A) and LF (0x0D)? is
is_a_parent (*tag*: *STRING*): *BOOLEAN*
-- True if *tag* is a container (parent) at the current level.
-- That can be a just started tag, or a tag higher up.
is_fragment: *BOOLEAN*
-- True if the XML document being created is a fragment.
is_header_written: *BOOLEAN*
-- True if the XML header is written, or if this is a fragment.
is_started (*tag*: *STRING*): *BOOLEAN*
-- True if this tag has just been started.
is_tag_started: *BOOLEAN*
-- True if one or more tags have been opened.
is_valid_attribute_name (*attribute*: *STRING*): *BOOLEAN*
-- Return True if this is a valid attribute name.
unfinished_xml: *STRING*
-- The *xml* in progress.
as_string: *STRING*
-- The result as plain STRING.
as_uc_string: *UC_STRING*
-- The result as Unicode string, i.e. UC_STRING.
feature(*s*) **from** *EPX_XML_WRITER*
-- Influence state

clear
-- Start fresh.
feature(*s*) **from** *EPX_XML_WRITER*
-- Commands that expand *xml*
add_header (*encoding*: *STRING*)
-- Add the XML header, document is encoded in
-- *encoding*. Making sure this encoding is followed, is the
-- responsibility of the client.
add_header_iso_8859_1_encoding
-- Document is iso-8859-1 encoded.
add_header_utf_8_encoding
-- Document is utf8 encoded.
add_data (*data*: *STRING*)
-- Write data in the current tag.
-- Invalid characters like < or > are quoted.
-- Use *add_raw* if you dont want quoting.
add_entity (*name*: *STRING*)
-- Write entity *name*.
add_raw (*raw_data*: *STRING*)
-- Write *data* straight in the current tag, meta characters
-- are not quoted, control characters are not checked, etc.
add_system_doctype (*root_tag*, *system_id*: *STRING*)
-- Add a <!DOCTYPE element.
-- Only allowed when no tags have been written.
add_tag (*tag*, *data*: *STRING*)
-- Shortcut for *add_tag*, *add_data* and *stop_tag*.
get_attribute (*attribute*: *STRING*): *STRING*
-- Get contents of attribute *attribute* for
-- current tag. *attribute* may include a name space.
-- Returns Void if attribute doesnt exist
put (*a*: *ANY*)
-- Write data within the current tag.
put_new_line
-- Add a new line in the current tag.
puts (*stuff*: *STRING*)
-- Write data within the current tag.
set_attribute (*attribute*, *value*: *STRING*)
-- Set an attribute of the current tag.
-- *attribute* must be name-space less, else use *set_ns_attribute*.
-- *value* may not contain an entity reference.
-- As the attribute is not immediately written, make sure
-- *attribute* and *value* do not change (ie are cloned or
-- immutable).
set_a_name_space (*a_prefix*, *a_uri*: *STRING*)
-- Define a name space.
-- As the attribute is not immediately written, make sure
-- *a_prefix* and *a_uri* do not change (ie are cloned or

```
        -- immutable).
    set_default_name_space (uri: STRING)
        -- Set the default name space.
    set_ns_attribute (name_space, attribute, value: STRING)
        -- Set an attribute of the current tag.  value may not
        -- contain an entity reference. name_space is the optional
        -- prefix to be used, not the actual URI.
        -- As the attribute is not immediately written, make sure
        -- name_space, attribute and value do not change (ie
        -- are cloned or immutable).
    start_ns_tag (name_space, tag: STRING)
        -- Start a new tag in the given name_space. name_space is
        -- a prefix only, not the actual URI. If name_space is Void
        -- or empty, the tag will not get a prefix.
        -- As the tag is not immediately written, be sure that tag
        -- does not change (ie is cloned or immutable) if
        -- name_space is Void or empty.
    start_tag (tag: STRING)
        -- Start a new tag.
        -- As the tag is not immediately written, make sure tag
        -- does not change (ie is cloned or immutable).
    stop_tag
        -- Stop last started tag.
feature(s) from EPX_XML_WRITER
    -- Quote unsafe characters
    replace_content_meta_characters (s: STRING)
        -- Replace all characters in s that have a special meaning in
        -- XML. These characters are < and &.
        -- For compatibility with SGML a "]]>" should be written as
        -- "]]&gt;", but I see no reason to be compatible with SGML
        -- these days, so we dont check for that.
feature(s) from EPX_XML_WRITER
    -- Comments
    start_comment
    stop_comment
invariant
    -- lower_a_code_definition: lower_a_code = (a).code
    -- Same thing for all other codes.
    -- (see "note" in indexing clause.)
    accessing_real_singleton: security_is_real_singleton;
    my_xml_not_void: my_xml /= Void;
    same_size: attributes.count = values.count;
    has_tag_stack: tags /= Void;
    fragment_has_no_header: is_fragment implies is_header_written;
    values_not_void: values /= Void;
    attributes_not_void: attributes /= Void;
```

```
    every_attribute_has_a_value: attributes.count = values.count;
end of EPX_XML_WRITER
```

F.5 EPX_XHTML_WRITER

class *interface EPX_XHTML_WRITER*
creation
make
-- Create an XML document with initial capacity of 1024 characters.
make_with_capacity (*a_capacity*: *INTEGER*)
-- Create an XML document with initial capacity of
-- *a_capacity* characters.
make_fragment
-- Create an XML fragment (document without header) with
-- initial capacity of 1024 characters.
make_fragment_with_capacity (*a_capacity*: *INTEGER*)
-- Create an XML fragment (document without header) with
-- initial capacity of *a_capacity* characters.
feature(*s*) **from** *EPX_XHTML_WRITER*
-- overrule some xml stuff
new_line_after_closing_tag (*a_tag*: *STRING*)
-- Outputs a new line, called when *a_tag* is closed
-- can be overridden to start a new line only occasionally
-- For XHTML documents a new line is treated as a single
-- space, so it can influence layout.
new_line_before_starting_tag (*a_tag*: *STRING*)
-- Outputs a new line, called when *a_tag* is about to begin.
feature(*s*) **from** *EPX_XHTML_WRITER*
-- doctype
doctype
doctype_frameset
-- Output will be frame-based
doctype_strict
-- Output will be strict XHTML
doctype_transitional
-- Output will be transitional XHTML
feature(*s*) **from** *EPX_XHTML_WRITER*
-- page
b_html
-- start html page
e_html
feature(*s*) **from** *EPX_XHTML_WRITER*
-- header
meta_refresh_other (*time*: *INTEGER*; *url*: *STRING*)
b_head
e_head
title (*a_text*: *STRING*)
feature(*s*) **from** *EPX_XHTML_WRITER*
-- body
b_body

e_body
feature(*s*) **from** *EPX_XHTML_WRITER*
--section headers
h1 (*header_text*: *STRING*)
feature(*s*) **from** *EPX_XHTML_WRITER*
-- paragraph
br
-- break, the Appendix C.3 way
b_p
e_p
p (*par*: *STRING*)
feature(*s*) **from** *EPX_XHTML_WRITER*
-- layout
b_tt
-- teletype writer font
e_tt
feature(*s*) **from** *EPX_XHTML_WRITER*
-- link
b_a (*href*: *STRING*)
e_a
a (*href*, *s*: *STRING*)
feature(*s*) **from** *EPX_XHTML_WRITER*
-- rules
hr
-- horizontal rule
feature(*s*) **from** *EPX_XHTML_WRITER*
-- white space
nbsp
-- non breaking white space
feature(*s*) **from** *EPX_XHTML_WRITER*
-- verbatim
b_pre
e_pre
feature(*s*) **from** *EPX_XHTML_WRITER*
-- tables
b_table
e_table
b_tr
e_tr
td
-- an empty cell
b_td
e_td
th
-- an empty cell
b_th
e_th

feature(*s*) **from** *EPX_XHTML_WRITER*
-- forms
standard_encoding: *STRING*
plaintext_encoding: *STRING*
multipart_encoding: *STRING*
b_form (*method, action*: *STRING*)
b_form_get (*action*: *STRING*)
b_form_post (*action*: *STRING*)
e_form
b_input (*type, name*: *STRING*)
e_input
button_hidden (*name, value*: *STRING*)
b_button_submit (*name, value*: *STRING*)
e_button_submit
b_button_reset
e_button_reset
button_reset
b_checkbox (*name, value*: *STRING*)
e_checkbox
b_radio (*name, value*: *STRING*)
e_radio
b_select (*name*: *STRING*)
e_select
option (*text*: *STRING*)
selected_option (*choice*: *STRING*)
b_textarea (*name*: *STRING*)
-- multiline input
e_textarea
input_text (*name*: *STRING*; *size*: *INTEGER*; *value*: *STRING*)
-- singleline input
feature(*s*) **from** *EPX_XHTML_WRITER*
-- CSS style sheet support
b_style
e_style
set_class (*name*: *STRING*)
-- set attribute class
invariant
-- lower_a_code_definition: lower_a_code = (a).code
-- Same thing for all other codes.
-- (see "note" in indexing clause.)
accessing_real_singleton: *security_is_real_singleton*;
my_xml_not_void: *my_xml* /= *Void*;
same_size: *attributes.count* = *values.count*;
has_tag_stack: *tags* /= *Void*;
fragment_has_no_header: *is_fragment* **implies** *is_header_written*;
values_not_void: *values* /= *Void*;
attributes_not_void: *attributes* /= *Void*;

```
    every_attribute_has_a_value: attributes.count = values.count;
end of EPX_XHTML_WRITER
```

G
Short (flat) listing of Single Unix Specification bonus classes

In this chapter:

Classes in this appendix are based on the Single Unix Specification. Some of them might also be available under Windows.

G.1 *EPX_HTTP_10_CLIENT*

```
class interface EPX_HTTP_10_CLIENT
creation
    make (host_name: STRING)
        -- Prepare for request to host_name.
    make_from_port (host_name: STRING; port: INTEGER)
        -- Prepare for request.
        -- Use port is 0 to use the default port (80).
    make_from_host (a_host: SUS_HOST)
        -- Prepare for request to resolved a_host. If port is 0,
        -- the default port is taken, else the port can be overruled.
    make_from_host_and_port (a_host: SUS_HOST; port: INTEGER)
        -- Prepare for request to a_host. If port is 0, the
        -- default port is taken, else the port can be overruled.
feature(s) from EPX_HTTP_10_CLIENT
    -- Client http version
    client_version: STRING
        -- Clients version of the http protocol
feature(s) from EPX_HTTP_10_CLIENT
    -- Requests
    get (path: STRING)
        -- Send GET request to server.
    head (path: STRING)
        -- Send HEAD request to server.
        -- path should not include http: and the host name, only
        -- the page that is requested. Any query and fragment parts are ok.
    options (path: STRING)
```

```
        -- Get server options. Path is required when the request is
        -- being made to a proxy.
feature(s) from EPX_HTTP_10_CLIENT
    -- Fields that are send with a request if set
    user_agent: STRING
        -- Identification of client program.
        -- Common examples are:
        --    Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
        --    Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.0.0) Gecko/20020529
        --    Microsoft Internet Explorer
    set_user_agent (value: STRING)
        -- Set the client identification.
feature(s) from EPX_HTTP_10_CLIENT
    -- Response
    body: EPX_MIME_BODY_TEXT
        -- Return body as text, if applicable, else Void.
    fields: DS_HASH_TABLE[EPX_MIME_FIELD,STRING]
        -- Header fields of response.
    is_response_ok: BOOLEAN
        -- Does the returned response_code indicate success?
    part: EPX_MIME_PART
        -- The entire parsed MIME message
    c: CHARACTER
    read_response
        -- Read entire resonse, parse while reading.
    response_code: INTEGER
    response_phrase: STRING
    server_version: STRING
        -- Set by read_response.
feature(s) from EPX_HTTP_10_CLIENT
    -- Individual response fields, Void if not available
    location: STRING
invariant
    accessing_real_singleton: security_is_real_singleton;
    host_found: host /= Void and then host.found;
    have_address: sa /= Void;
end of EPX_HTTP_10_CLIENT
```

G.2 ULM_LOGGING

This class depends on Standard C only. It is the *EPX_LOG_HANDLER* that is platform specific. e-POSIX provides implementations of this class for Unix through syslog and for Windows through the NT event log.

```
class interface ULM_LOGGING
creation
    make (a_handler: ULM_LOG_HANDLER; a_program_name: STRING)
        -- Start logging for program. The host name is derived from
        -- an OS specific call through a_handler.
feature(s) from ULM_LOGGING
    -- Log methods
    log_error (level: INTEGER; subsystem: STRING; error_number: INTEGER; error_message: STRING)
        -- Useful for logging errors.
    log_event (level: INTEGER; subsystem: STRING; fields: ARRAY[ULM_FIELD])
        -- Log event, consisting of one or more fields. It is the
        -- responsibility of the client to make sure the values are
        -- proper for each field.
        -- This function adds any ULM required field if not present.
        -- subsystem, if present is appended with a dot to
        -- program and written in the "PROG" field.
        -- DATE is logged in GMT.
    log_single_field (level: INTEGER; subsystem, field_name, value: STRING)
        -- Log value for field_name. value will be properly
        -- quoted if necessary. value should be in the proper
        -- format for field_name.
        -- This function adds any ULM required field.
        -- subsystem, if present is appended with a dot to
        -- program and written in the "PROG" field.
        -- in the "PROG" field.
        -- DATE is logged in GMT.
    log_message (level: INTEGER; subsystem, value: STRING)
        -- Log a simple message with the MSG field.
        -- This function adds any ULM required field.
        -- subsystem, if present is appended with a dot to
        -- program and written in the "PROG" field.
        -- DATE is logged in GMT.
feature(s) from ULM_LOGGING
    -- Queries
    is_valid_field_name (field_name: STRING): BOOLEAN
        -- Returns True if field_name is valid according to ULM spec.
        -- Basically it should consist of one or more letters and have
        -- no spaces.
    is_valid_partial_field_list (fields: ARRAY[ULM_FIELD]): BOOLEAN
        -- Contains True if fields contains at least one item, and
        -- if every item in fields is not Void and if fieldsdoes
        -- not contain a duplicate field and if fields does not
```

-- contain the LVL field.

feature(*s*) **from** *ULM_LOGGING*

-- Standard field names

lvl: *STRING*

-- Importance and category of the ULM.

host: *STRING*

-- Name of software component which issues the ULM.

prog: *STRING*

-- Name of the software component which issued the ULM.

date: *STRING*

-- Instantaneous date of the event.

lang: *STRING*

-- Language used for text fields. Default is english (EN).

dur: *STRING*

-- Indicates duration (in seconds) of the event.

ps: *STRING*

-- Process id which issued the ULM.

id: *STRING*

-- System reference to the concerned document.

src_ip: *STRING*

-- The IP number of the source host.

src_fqdn: *STRING*

-- Fully qualified Domain Name for the source host.

src_name: *STRING*

-- Generic name qualifying the source.

src_port: *STRING*

-- Port number for TCP, UDP or other protocol.

src_usr: *STRING*

-- User name or user id.

src_mail: *STRING*

-- Email address.

dst_ip: *STRING*

-- The IP number of the destination host.

dst_fqdn: *STRING*

-- Fully qualified Domain Name for the destination host.

dst_name: *STRING*

-- Generic name qualifying the destination.

dst_port: *STRING*

-- Port number for TCP, UDP or other protocol.

dst_usr: *STRING*

-- User name or user id.

dst_mail: *STRING*

-- Email address.

rel_ip: *STRING*

-- The IP number of the proxy/relayer host.

rel_fqdn: *STRING*

-- Fully qualified Domain Name for the proxy/relayer host.

rel_name: *STRING*
-- Generic name qualifying the proxy/relayer.
rel_port: *STRING*
-- Port number for TCP, UDP or other protocol.
rel_usr: *STRING*
-- User name or user id.
rel_mail: *STRING*
-- Email address.
vol: *STRING*
-- Volume (number of bytes) sent and received from the source
-- point of view.
vol_sent: *STRING*
-- Volume (number of bytes) sent from the source point of view.
vol_rcvd: *STRING*
-- Volume (number of bytes) received from the source point of view.
cnt: *STRING*
-- Count (of articles, files, events) sent and received from
-- the source point of view.
cnt_sent: *STRING*
-- Count (of articles, files, events) sent from the source
-- point of view.
cnt_rcvd: *STRING*
-- Count (of articles, files, events) received from the
-- source point of view.
prog_file: *STRING*
-- Name of the program source file from which the ULM was generated.
stat: *STRING*
-- State or status of the designed process. Possible values
-- for this field may include "Failure", "Success", "Start",
-- "End".
tty: *STRING*
-- Users physical connection to the host.
doc: *STRING*
-- Name of accessed document like the path of an ftp file,
-- the name of a newsgroup, or the non-host part of an URL.
prot: *STRING*
-- Protocol used.
cmd: *STRING*
-- Issued command.
msg: *STRING*
-- The only field which should contain arbitrary data.
feature(*s*) **from** *ULM_LOGGING*
-- Public state
host_name: *STRING*
-- Name of the host which issues the ULM.
program_name: *STRING*
-- Name of the software component which issues the ULM.

```
invariant
    log_level_text_lower_index_ok: log_level_text.lower = emergency;
    log_level_text_upper_index_ok: log_level_text.upper = debugging;
    accessing_real_singleton: security_is_real_singleton;
    handler_not_void: handler /= Void;
    host_name_not_empty: host /= Void and then not host.is_empty;
    program_name_not_empty: program_name /= Void and then not program_name.is_empty;
    have_my_date: my_date /= Void;
    have_my_host: my_host /= Void;
    have_my_prog: my_prog /= Void;
    have_my_lvl: my_lvl /= Void;
end of ULM_LOGGING
```

To do

EPX_FILE_SYSTEM

1. Make *EPX_DIRECTORY*.

STDC_FILE

1. read_integer, read_double, read_boolean should perhaps be different for the binary or text files. Now they're satisfy the mico/e definition, so useful for text files only.

STDC_LOCALE_NUMERIC

1. Complete the list of properties

STDC_PATH

1. make some escape char functionality with '%' or so.

POSIX_STATUS

1. return STDC_TIME instead of unix time
2. Not all stat member fields are currently available.

STDC_TIME

1. Add elapsed seconds

POSIX_EXEC_PROCESS

1. turn off Eiffel exception handling after the final execvp, else you get back signals not captured by child process as your signals, or so it seems (or perhaps you're killing the Eiffel process, but not the subprocess it generated??)

   Killing subprocesses works sometimes, but not always.

   Remove exception handling just before execvp?
2. how about capture to /dev/null?

3. can we capture i/o for every forked process? If so, move this code to POSIX_FORK_ROOT.
4. Perhaps option to influence environment variables to pass to subprocess?

POSIX_FILE_DESCRIPTOR

1. possible to open exclusively and so?
2. complete support for nonblocking i/o.

POSIX_MEMORY_MAP

1. Cannot change protection.
2. No locking.

POSIX_SEMAPHORE

1. not valid for named semaphore I think.
2. have to add various close/unlink functions.

POSIX_SIGNAL

1. Add synchronous waiting for signals like `sigwait`.
2. (Re)enable sending Eiffel exception on signal? i.e. set_exception_handler or so.
3. Resend signal as Eiffel exception in signal handler.

MQUEUE

1. Not in the free unices at this moment. Maybe have to get a copy of Solaris x86??

Security

Add base security class that specifies programs intent. Default is to allow anything, but security can be tightened:

1. Call to `open`or `creat`(used?), use real user id, not effective user id.
2. Assume we're free from buffer attacks if preconditions are enabled.
3. exec/system call only allowed when effective user is not root, unless otherwise specified. Or exec only allowed for specific files.

4. Protect against writing specific files/directories. Perhaps substitute vulnerable filenames for other ones.

5. Emulate atomic calls. Or add atomic `access`and `open`call. Shouldn't be done by setting su??

6. When appending/writing to files, check if symbolic link.

7. *ABSTRACT_FILE_SYSTEM.force_remove_directory* is potentially unsafe because it follows links so it can be used to destroy things not under that directory.

8. remove tmpnam function.

9. Make sure the once functions in STDC_BASE are called from within the security initialization, so they're allocated and do not generate an out-of-memory exception themselves.

Idea from 'Remediation of Application Specific Security Vulnerabilities at Runtime' article in IEEE Computer sep/oct 2000.

Windows code

1. chmod also available on Windows.

2. Add permissions to status: read/write.

3. set_binary_mode should do something for the posix factory, i.e., when compiling with cygwin. Perhaps separate *CYGWIN_API* or so in POSIX dir with the window specific stuff.

   Currently cygwin uses text mode for file descriptors, the windows variant uses binary.

4. utime can be supported by using SetFileTime.

Other

1. remove ugly const_ prefix from constants. Uppercase should be good enough.

   Almost done, only const_EOF remains, not easy to replace perhaps.

2. Compare POSIX_SIGNAL with ISE UNIX_SIGNAL: They have an is_caught function, useful? Means this signal generates an exception.

Known bugs

- not for every *STDC_BASE.raise_posix_error* the error code is set probably.

- does STRING_HELPER leak memory in to_external? How is memory used for these conversions being freed? Is memory used there?

- If a child process is signalled (terminated), the function *POSIX_FORK_ROOT.is_terminated_normally* sometimes returns True.

Bibliography

1 (1996). *System Application Program Interface (API) [C Language]*, volume Part I of *Information technology – Portable Operating System Interface (POSIX)*. ANSI/IEEE, 1996 edition.

2 (1991). *The Standard C library*. Prentice Hall.

3 (1994). *POSIX programmer's guide*. O'Reilly & Associates.

4 Stevens, W. R. (1998). *Unix network programming*. Prentice Hall.

5 Meyer, B. (1997). *Object-Oriented Software Construction*. Addison Wesley, 2nd edition.

6 Hatton, L. (2001). Exploring the role of diagnosis in software failure. *IEEE Software*.

7 Whittaker, J. A. (2001). Software's invisible users. *IEEE Software*.

Index