Testing the POSIX Socket Packages

Bruce Herring
TR-000401
Department of Computer Science
Florida State University
Tallahassee, FL 32312

April 25, 2000

Abstract

A series of tests have been created for the sockets portion of the POSIX Ada binding standard (POSIX.5c). The goal of the design and implementation of the tests was twofold. The first objective was to design a series of tests that would throughly test the conformance of any POSIX.5c implementation to the standard. The second objective was to test the Florist implementation of POSIX.5c for errors. The two objectives are considered to be of equal importance.

1 Introduction

This document describes the design, implementation, and execution of a series of tests designed to test implementations of the socket portions of the POSIX-Ada binding standard (POSIX.5c). The tests are needed to gauge the conformance of a POSIX.5c sockets implementation with respect the POSIX.5c standard. The development of the tests was done using the latest version of the Florist implementation of POSIX.5c. This implementation has never been rigorously tested with respect to sockets. As a second, but not secondary, goal, any errors found in any of the Florist files while writing the tests was corrected.

This document is comprised of five sections. The first section is the introduction. The second section discusses the requirements for the tests. Section three describes the issues involved in designing the tests. In the fourth section, an explanation of the implementation of the tests is given along with a list of all performed tests. The fifth section discusses possible future extensions to the tests. Section six contains comments intended to help future modifiers of the Florist implementation. Appendix A contains a listing of all the changes made to `POSIX.Sockets` and `POSIX.Sockets.Internet` packages. Finally, Appendix B is a copy of the code for all the tests.

1.1 Background

The POSIX.5c standard was created in 1998. This standard was designed to provide an Ada counterpart to the C standard POSIX.1g. Florist is an implementation of the POSIX.5b standard that is being modified to be an implementation of the POSIX.5c standard as well. While the POSIX.5b portions have been thoroughly tested the POSIX.5c portions haven't. POSIX.5c is an

IEEE standard and cannot be changed without the approval of an IEEE committee. Therefore any error found in the standard may be changed in the Florist implementation, but these changes in no way affect the actual standard.

1.2 Terminology

Ada was used to implement all the tests so some Ada terminology is required. However, since the Ada language was not the focus of the project the amount of Ada terms used will be kept to a minimum. Listed below are definitions for the most commonly used Ada terms written in a terms of the C++ language. Also included are the full IEEE names for the commonly referred to standards.

Package: The closest C++ equivalent to a package is the `class` type. While they are different this equivalence should be enough to understand its usage for this project.

Generic: A generic is loosely equivalent to a template in C++.

POSIX.1g: IEEE Standard 1003.1g-1996 [1]

POSIX.5C: IEEE Standard 1003.5c-1998 [2]

2 Requirements

The intended purpose of the tests is to determine the completeness and correctness of an implementation of the sockets portion of the POSIX.5c standard. However, the Sockets portion of the Florist implementation of POSIX.5c has never been thoroughly tested. Therefore a second objective of the tests was to correct any errors in the sockets portion of the Florist packages.

This section discusses the requirements for writing the tests that test the conformance of the sockets portion of a POSIX.5c implementation. It also describes the requirements for testing the sockets portion of the Florist implementation. Finally some of the major differences between the goals of the two areas of testing will be discussed.

2.1 Conformance Tests

POSIX.5c is a standard that can be used on many different computer architectures. Different architectures, however, do not always support the same operations. Even if the same operation is supported the error messages associated with the operation may be different. A test for conformance determines which aspects of the standard are supported on a given system. It also checks the error messages of any supported operations against the expected error messages.

Testing for conformance requires several aspects of the packages to be tested. The exact pieces that must be tested are dictated by the Test Development Plan [3]. The portion of the Plan that deals with what tests were created is given below.

> For each POSIX.5b package, there is at least one standard-driven test, whose name corresponds to the section where the package is defined. For example, since the package `POSIX_Timers` is defined in Section 14.1 of POSIX.5b, there is a standard-driven test named `p140100`. The procedure for writing such a test is as follows:

1. Read each the section of the standard that defines the package and its semantics, sequentially. For each subsection of the standard, create one or more subsections of the test, preserving the order. Document the correspondence of the test code to the numbered subsections of the standard.

   For example, Section 14.1.4 of POSIX.5b defines the functions `Set_Time`, `Get_Time`, and `Get_Resolution`. Tests for these functions should be grouped together, following the tests for Section 14.1.3 and preceding the tests for Section 14.1.5.

   > NOTE: Sometimes the order of presentation in the standard is not suitable for testing. For example, it may be necessary to use operations defined later in a section, in order to check an assertion about an operation defined earlier in the same section. Under such circumstances it is acceptable to reorder the sections of the test.

2. For each constant declaration, test the following:

   (a) Use the constant at least one, to verify that it is declared.

   (b) If several constants of the same type are required to have distinct values, check that they are distinct. The following is an example of how this can be checked for the constants of the type `POSIX.Option_Set`.

```
Test ("Option_Set type");
declare
   A : constant array (1 .. 31) of Option_Set :=
     (Option_1,
      Option_2,
      Option_3,
      Option_4,
      Option_5,
      Option_6,
      Option_7,
      Option_8,
      Option_9,
      Option_10,
      Option_11,
      Option_12,
      Option_13,
      Option_14,
      Option_15,
      Option_16,
      Option_17,
      Option_18,
      Option_19,
      Option_20,
      Option_21,
      Option_22,
      Option_23,
      Option_24,
      Option_25,
      Option_26,
      Option_27,
      Option_28,
      Option_29,
      Option_30,
      Option_31);
begin
   for I in Arrange loop
      for J in Arrange loop
         Assert (A (I) /= A (J) or I = J,
```

```
                "option set constants are distinct");
          end loop;
       end loop;
    exception when E : others => Fail (E);
    end;
```

3. For each type declaration, use the type at least once, to verify that it is declared.
4. For each literal of an enumeration type, check that the position is correct. The following is an example of how the three literals of the type POSIX.Blocking_Behavior are tested.

```
Assert (Tasks = Blocking_Behavior'First and
  Blocking_Behavior'Pos (Program) = 1 and
  Special = Blocking_Behavior'Last,
  "Blocking_Behavior literals");
```

5. For each subtype with a specified upper or lower bound, check that the bound is the one specified. The following is an example of how the lower bound of type POSIX.IO_Count_Maxima is tested.

```
Assert (IO_Count_Maxima'First = 32767, "IO_Count_Maxima'First");
```

6. For abstract types with matched Get_*X* and Set_*X* operations, use the Set_*X* operation to set a value, then check that the Get_*X* operation returns the value that was just set. Look out for anomalous cases, such as setting the baud rate of a terminal device to zero, where a special value for the Set_*X* operation means to set the value to a default and the Get_*X* operation returns the actual default value.
7. For abstract types whose default initial values are specified by the standard, declare a variable of the type without any initial value expression and then check that the initial value satisfies the requirement of the standard.
8. For each subprogram declaration check the following, in decreasing order or priority:
   (a) Call the subprogram with valid parameters to verify that it is declared and can be called successfully. Verify that the effect of the call is consistent with the standard.
   (b) If the subprogram has default parameters call it using the defaults. If possible, verify that the correct default values are used.
   (c) For each error condition that is required to be detected set up a situation that causes the error, if possible, and verify that POSIX_Error is raised with the correct error code.
   (d) For each error condition that is optionally detected set up a situation that causes the error, if possible, and verify that if POSIX_Error is raised it has the correct error code.
   (e) If there is optional functionality or a capacity limit associated with a subprogram test that the support is consistent with what is reported (e.g. by the packages POSIX_Limits, POSIX_Options, POSIX_Configurable_System_Limits, or POSIX_Configurable_File_Limits). For capacity limits, also report the actual and any inferred limits.
   (f) If there is a natural way to extract performance information as a by-product of testing report the observed information using a call to the Comment procedure

(so it will only report the information with the `-v` option). For example, as a by-product of testing interval timers, `p140100` optionally outputs the requested and observed time delays.

9. If an interface overlaps functionally with an Ada language-defined feature, check for consistency and interoperability. For example, it should be possible to create a file using the POSIX.5 interfaces and then read it later using the standard Ada `Text_IO` interfaces. Similarly, it should be possible to pass command-line arguments via the POSIX.5 interface to `Start_Process`, and then retrieve the arguments via the `Ada.Command_Line` interface.
10. Look for English statements of requirements (e.g. those with the verb "shall"), and if they state a requirement not already covered by any of the above, create an additional check for the requirement. Provide a comment explaining what is being checked.

A failure in the conformance tests indicates that either an operation is not supported or its error messages does not conform with the accepted standard error messages for that operation.

2.2 Florist Tests

While the latest version of the POSIX.5c standard has existed for over two years not all of the Florist implementation has been thoroughly tested. The socket packages happen to be some of the packages that fall into this category. While creating the completeness tests it is possible, and expected, that some errors in Florist will be detected. When found these errors should be corrected. Some possible sources of errors and common solutions are given below.

The Florist implementation of the POSIX.5c standard is very large and complicated. There are several programs that are used to configure the environment so compilation is possible on different operating systems. Some aspects of the implementation that the configuration programs control include the values of constants, which constants are supported, compilation order, and the creation of some core packages. Any one of these areas could have an error that was never detected before because the socket packages are the only ones that depend on it. When erroneous constants are found the correct values, as dictated by the POSIX.5c standard, should be inserted. If missing constants occur they should be added but only in such a way that allows the configuration tool to verify that they are indeed supported by the system.

Since the socket packages are the ones being tested they are clearly where most errors will originate. Errors in the these packages can come from several different sources. Occasionally there will be differences between the POSIX.1g standard and the POSIX.5c standard. In these cases the Florist implementation should be changed to conform with the POSIX.1g standard. Some error messages that were expected in the POSIX.1g standard cannot be recreated in the POSIX.5c standard. This is due to the differences between ANSI C and Ada. When this occurs the tests for the error message in question should not be implemented. Due to the looseness of the POSIX.1g standard in the area of generated error codes, it is possible that one error will occur when a different one was expected. For this situation only the tests for the errors that are expected and are generated should be created. In rare occasions there may be pieces of the POSIX.1g standard that were never placed in the POSIX.5c standard. As long as the desired result can be created in the Florist implementation the Florist files should be changed to match the POSIX.1g standard. Constants may not agree with what is specified. In this case the constants in the socket packages should be changed to match the documented ones.

Aside from differences between the two standards there are several other ways errors can creep in. Some secondary files may contain errors that need to be corrected. Optional pieces may not have the appropriate functions and constants created for them. Finally, there are always logic and syntax errors that were never corrected before.

Whenever a problem with a test cannot be resolved by one of the above solutions it should be removed and a comment placed in the code. The comment should indicate what test was attempted, how it failed, and any insight the programmer had on the situation.

2.3 Difference in Tests

The same code is used to perform two different functions, both conformance testing and testing the Florist implementation. Sometimes it can be difficult to determine in what capacity a test is being used. Some general guidelines are presented here to help simplify that task.

In general if the creation of a test causes a change in any of the Florist implementation packages then it is being used as an implementation test. A failure in an implementation test should be corrected. By the completion of all the tests there should be no failures due to implementation issues.

Somewhat in opposition to implementation tests are the conformance tests. Errors in conformance tests should not be fixable by changing the code in the Florist implementation packages while maintaining the correctness of the standard. Conformance errors are indicative of non-support or behavior that is inconsistent with the standard. The goal of the conformance tests is to determine a systems level of conformity to the standard. With this in mind, errors are expected and should be viewed as a necessary part of testing.

3 Design

The design phase of any software project is intended to create a detailed outline that will guide the project's implementation. The design of the socket tests is no different. Since the tests are part of a larger suite of tests they will need to conform to the style used in the other tests. Occasionally the guidelines do not contain the level of detail that is desired. In these situations new, or more specialized, design guidelines must be established.

With the socket tests design there were two different objectives had to be combined into one implementation. These objectives consist of testing a system for the inclusion of all aspects of the POSIX.5c standard as well as testing the sockets portion of the Florist implementation. The design for conformance testing was thoroughly laid out. However, the design for the Florist implementation testing had not been specifically specified. Both design objectives will be discussed in the following paragraphs.

3.1 Style of Completeness Tests

The style used in all the conformance tests for the POSIX.5c standard is documented in the Testing Development Plan [3]. The portion of the Plan that deals with style is recounted below.

1. Follow the layout style rules enforced by the GNAT compiler's "`-gnatg`" option. These cover indentation, the use of spaces, and consistency of usage of upper and lower case letters in identifiers.

2. Follow the POSIX.5b standard's rules for when to use upper and lower case letters in identifiers. That is, the first letter of each word of an identifier is in uppercase, and succeeding letters are in lowercase, with underscores used to separate words. An exception is where an identifier includes a proper name (e.g. `POSIX`) or abbreviation (e.g. `ID` in `Process_ID`) where there is a well established convention that it appear in all capital letters.

3. Use a block comment starting with "`-- ....`" to explain a need for further work, e.g. a group of cases for which more checks should be added.

4. Use a block comment starting with "`-- ????`" to document a problem or question that needs to be resolved.

5. Use a comment of the following form to document in English an abstract test assertion.

```
------------------------------------------------------------
--  An English-language assertion is a statement that should
--  be true, based on reading the standard, and for which the
--  following block of code performs a check.  This may be a
--  sentence or a paragraph.  It should be stated in as abstract
--  terms as possible, and should ideally make reference
--  [in brackets] to a section or sections of the standard.
```

   Generally, there should be such a comment for each major block of code (usually coded as a `begin ... end` block) in a test program.

   Such assertions may include bits of relevant text taken from the standard, but should focus on the functionality being tested in a specific block of code. (They should not be repetitious, or contain text from the standard that is not locally relevant.)

6. Each test program should be a self-contained Ada procedure, if possible.

   > Exceptions to this are (1) where there is a need for a child process or (2) where several tests can share some common code, as in the `p9900` series of tests.

7. For standard-driven tests start with a single program for each package, whose number starts with the section and subsection numbers and ends in `00`. For example, the name of the standard-driven test program for package `POSIX_Timers`, which is defined in Section 14.1 of the POSIX.5b document, is `p140100`.

   Additional standard-driven tests should be added to cover situations where a check requires child processes or where failure of a check is likely to result in the test hanging or terminating prematurely so that other checks are not performed. The latter includes situations where the effect of an operation being tested is "unspecified" or "undefined". Also split tests to separate checks that rely on an optional feature or a feature that is not included in one of the real-time POSIX profiles.

8. For standard-driven tests avoid using POSIX packages other than the primary one being tested. If that package requires the use of other packages, use them, but only to the minimum degree needed.

9. Tests must be examples of *portable* conforming POSIX Ada applications. In general one should avoid using `Unchecked_Conversion` or other type-unsafe coding techniques in tests. However, for tests of some low-level interfaces, such as I/O using `Stream_Element_Arrays`, unchecked conversions are required. Likewise, tests of the memory mapping interfaces must use addresses. When using these "unsafe" interfaces one still must use them in a way that is portable.

10. Never put out any messages except via the package `POSIX_Report`. The only permitted use of the Ada language-defined I/O packages is to check for interoperability of Ada and POSIX I/O.

11. Avoid using the `Failure` procedure in `POSIX_Report`. Instead, use more specific procedures like `Assert` and `Unexpected_Exception`.

12. Every failure message, regardless of how it is generated, must contain a unique identifier of the form A*nnn*, where *nnn* is a 3-digit zero-filled number that is unique to the specific test file. Initially, the identifiers should be sequential within the text of the file where they appear, though this may change as the program is modified later. If a program requires more than one file, for files other than the main procedure append the name of the file to the identifier, e.g. `A034: p990000`. Examples of subprograms that may generate failure messages, and so require such a parameter, include `Assert`, `Unexpected_Exception`, `Check_Error_Code`, and `Optional`.

    These unique identifiers should eventually be frozen, once the test suite is under configuration control. If code is inserted during maintenance, try to insert it at the end of the program, so that the failure message sequence can be preserved. Better yet, put the new checks into a separate test program.

    The intent *behind these rules is to allow communication with users about test failure messages, even if they are using an older version of the given test, and to allow building a (stable) cross-reference between failure messages of the POSIX.5b tests and assertions in one of the sets of test assertions produced for the POSIX C-language interfaces.*

13. Whenever possible, avoid using new Ada'95 language features, that are not supported by Ada'83. For example, use `Text_IO` instead of `Ada.Text_IO`. The main exception permitted to this rule is the use of an exception identifier in exception handler choices. (In principle it should be possible to preprocess the tests to remove these usages for compilers that do not support this Ada'95 feature.)

14. Never use the child-package form of a POSIX package name. For example use "`with POSIX_IO;`" rather than "`with POSIX.IO;`".

15. Never use the package `Ada.Exceptions` directly in a test program; all usage should be encapsulated in the package `POSIX_Report`.

16. When checking for consistency of POSIX features with Ada'95 features, such as `Ada.Command_Line` and `System.Real_Time`, put these checks into separate test programs, so that the rest of the tests can still be compiled using an Ada'83 compiler (assuming the exception identification parameters are removed from the handlers of the tests and `POSIX_Report` is rewritten to removed references to features from the `Ada.Exceptions` package).

17. Separate capacity tests from basic functionality tests. Each interface should be tested first in a way that does not depend on more than the minimum portable values of implementation-dependent capacities (e.g. the number of open file descriptors). This does not rule out writing additional tests to check the actual capacities against the reported limits.

18. Each optional feature should be tested first independently, in every way that it can be tested without depending on other options, before it is tested in combination with other options.
19. The first-executed statement of each test program should be a call to `Header` with the name of the test. For example, the test `p140100` should start with the following call.

    ```
    Header ("p140100");
    ```

    There should be only one call to `Header` per test program execution. Try to avoid initializations with side-effects that produce output before the call to `Header`.
20. The last-executed statement of each test program should be a call to the procedure `Done`, for normal completion, or to the procedure `Fatal_Exception`, for completion via propagation of an exception to the end of the main procedure body. The following example shows how this relates to the overall structure of a test program.

    ```
    procedure p140100 is
       ...
    begin
       Header ("p140100");
       --------------------------------------------
       ...
       --------------------------------------------
       Done;
    when E : others => Fatal_Exception (E);
    end p140100;
    ```

21. Divide the program into subtests, documented using the `Test` procedure call. There should ordinarily be a call to `Test` for each “Synopsis” subsection of the standard. Include the subsection number in brackets at the end of the comment string. For example, the following call might be used for a block of code that tests Section 14.1.3.

    ```
    Test ("Resolution (Clock_Realtime) [14.1.3]");
    ```

22. Use horizontal lines of “-”, with one empty line before and after, to separate subtests visually.
23. Isolate subtests, using exception handlers. Failure of one subtest should not terminate a test program, For example, the following isolates the success/failure of the call to `Get_Resolution`.

    ```
    Test ("Resolution (Clock_Realtime) [14.1.3]");
    declare
       Resolution : Timespec;
    begin
       Resolution := Get_Resolution (Clock_Realtime);
    exception when E : POSIX_Error => Fail (E);
    end;
    ```

24. Use the `Assert` procedure to verify postconditions. For example, the following call to `Assert` verifies that the value returned by `Get_Resolution` is consistent with the clock resolution reported by the corresponding function in `POSIX_Limits`. Include as the second parameter to `Assert` a short string that uniquely identifies the call within the test program.

```
Resolution := Get_Resolution (Clock_Realtime);
Split (Resolution, S, NS);
Assert (Seconds_Of (S) = 0 and Nanoseconds_Of (NS)
  <= POSIX_Limits.Portable_Clock_Resolution_Minimum,
 "A030");
```

If there is additional information that might be useful in evaluating the failure and that is short enough to fit on the same line with the failure message, you may provide a slightly longer message, such as the following.

```
Assert (Seconds_Of (S) = 0 and Nanoseconds_Of (NS)
  <= POSIX_Limits.Portable_Clock_Resolution_Minimum,
 "A030: resolution =" & Image (Resolution));
```

Don't use `if` statements for checking assertions where a simple call to `Assert` can be made to work.

25. Use the `Expect_Exception` procedure (not `Assert (False,...)`) for locations that control should never reach because an exception should have been raised by the preceding operation.

26. Use the `Check_Error_Code` procedure to check that the error code is correct inside a handler for `POSIX_Error` when the test requires the exception to be raised with a specific error code.

27. Use the `Comment` procedure to provide output that might be helpful to a user in isolating a failure, or to provide useful performance information. Remember that output from `Comment` is only printed out if the test is run in verbose mode.

    ```
    Resolution := Get_Resolution (Clock_Realtime);
    Comment ("Clock_Realtime reported resolution", Resolution);
    ```

    Do not use `Comment` to report failures, since it does not increment the failure count and does not print out anything unless the test is run in verbose mode.

28. Beware of legal implementation options. Any test of a feature that depends on an option must distinguish failure of the test due to nonsupport of the option from a true failure. The procedure `Optional` is provided to simplify this. For example, in the following code one version of this procedure is used to verify that `POSIX_Error` is raised with error code `Operation_Not_Implemented` if and only if the Timers Option is not supported.

    ```
    begin
       Resolution := Get_Resolution (Clock_Realtime);
    exception
    when E1 : POSIX_Error =>
       Optional (Timers_Option, Operation_Not_Implemented, E1);
    when E2 : others => Fail (E2);
    end;
    ```

29. Avoid repeating large blocks of code, or comments. A warning sign is if you find yourself re-typing or using the text editor's cut-and-paste functionality to repeat text. If this happens, you should probably be using a subprogram or a loop. Be on the lookout for code that is replicated across many tests, for possible addition as subprograms of the `POSIX_Report` package.

30. When a test requires multiple processes, make the body of the child processes separate from the main program. Give the file containing the main procedure of the child process a name that is the same as that of the main procedure but with a single lowercase letter appended. For example, if `p120501.adb` requires a child process, the main procedure goes in file `p120501a.adb` and is named procedure `p120501a`.
31. Never use the unsafe process primitives (e.g. `Fork`) in tests of other packages. Instead, use the `Start_Process` operations.
32. Never have a process start up another copy of itself or write a loop or recursion containing code that starts up processes, since failure of such a test can be disastrous (it leads to rapid proliferation of processes, to the point the system may need to be rebooted).
33. Try to avoid having many different main procedures for child processes of a single test. When a test needs child processes that do different things, write one procedure for the child and use command-line arguments or environment variables to tell each child what to do. The `-child` n argument is parsed automatically by the `POSIX_Report` package and the value of the positive integer n is stored in the variable `Child` for convenience in passing the identify of each child to the child.
34. Use the procedure `Check_Child_Status` in `POSIX_Report` to check the termination status of a child process, whenever possible, rather than writing out specific code to do this for each test.
35. When a test requires library-level packages, give the packages names related to the main procedure which uses them. For example, if `p120501.adb` requires a special library level package it could be called `p120501b` or `p120501c`.
36. Be on guard for features that are allowed to vary between implementations. The tests must be written in a way that whether a test passes or fails does not depend on such features at all, or the implementation dependence must be isolated in the package `Test_Parameters`. The following are some specific examples of implementation dependences for which the tests need to be parameterized:

    - The form of valid shared memory object names. Any test that expects to use a shared memory object must use a valid name.
    - The form of valid message queue names. Any test that expects to use a message queue must use a valid name.
    - The form of a valid semaphore name.
    - Valid device names, including block devices and character special files.
    - Nonexistent file names.
    - Reserved signals that are not defined by the standard.
    - Signals for which the application cannot set the action.
    - Whether the signal mask is process-wide or per-task.
    - A suitable small time delay value. There are many places in tests where we need to force a task or process to give up the CPU to allow another to execute, but if we delay much longer than necessary the test may give the impression of hanging.

    During maintenance of the Test Suite, as new examples of needs for such implementation-specific parameterization are detected, the `Test_Parameters` package should be expanded to cover them.

3.1.1 Strictness of Guidelines

While the guidelines in the Test Development [3] plan are strictly stated, the exact level of conformity to the guidelines in the existing POSIX.5b tests are not quite as strict. Many of the older tests do not follow the guidelines in any discernible fashion. For the Socket tests the guidelines were followed except for the following two cases:

1. The format of the argument to the `Test` function. Instead of using a

   Function name (Error)

   format, the format

   Function name: Error

   was adopted. Part of the reasoning behind this change was to maintain uniformity with the notation used for error codes in multiple process tests. The rest was simply author's discretion and the lack of a consistent form of notation in the other tests with respect to this one area.

2. The tests for the `Send` and `Receive` subprograms involve several child programs. Each child process holds a complete test for a single `Send` subprogram. The current design calls for each `Receive` to be paired with a `Send` of the same type. For example, consider the following:

```
procedure Receive
    (Socket           : in  POSIX_IO.File_Descriptor;
     Buffer           : in  System.Address;
     Octets_Requested : in  POSIX.IO_Count;
     Octets_Received  : out POSIX.IO_Count;
     Options          : in  Message_Option_Set);
```

   is paired with

```
procedure Send
    (Socket           : in  POSIX_IO.File_Descriptor;
     Buffer           : in  System.Address;
     Octets_Requested : in  POSIX.IO_Count;
     Octets_Received  : out POSIX.IO_Count;
     Options          : in  Message_Option_Set);
```

   However, this is not required by the standard. Future additions to the tests may want to use one type of `Send` for several `Receives`. By having each `Send` be in a separate process this can be done very efficiently. If every `Send` were in a single process such a test would require additional resources to load pieces of data that would never be used.

3.1.2 Further Compactness Guidelines

While Section 2.1 briefly discusses the compactness of code there are some areas that need stronger guidelines. Compact code is code that has made maximum use of functions and generics. If neither of these two language mechanisms are used the code for a single series of tests might be 3, 4, or more times larger than necessary. Functions can make one piece of code reusable, allowing only a small function call to be used instead of a large section of code. Generics take this idea one step further. They allow for the types of input parameters to a generic to be changed as well as the values. This means one piece of code can be used in many places even if the parameter types vary. However, generics should not be used when a simple function would work just as well.

The most common place where functions will greatly decrease the amount of code written is with error messages. One Socket subprogram may need to be tested for several error messages. The difference between tests is very small. A function should be written if there are more than one or two error cases. The function controls the call to the socket subprogram being tested. It also handles the the test to make sure an error occurred, includes handlers that will catch the expected exceptions and check there associated error code, and handles any unexpected errors. The parameters for the function should allow each error case to be set up external to the function. By changing the parameters the appropriate error will be caused inside the function. In most circumstances the same function that tests for errors should also be used to test for the non-erroneous case.

Generics should be used when there are several socket subprograms that all perform similar actions, have the same number of inputs, but some or all of the argument types differ. A common example are all the functions that set and get socket options. Other such cases may appear as well. The generic should be able to not only handle the testing of the correct cases but also any error cases as well. Aside from handling multiple argument type generics should function similar to the functions described in the previous paragraph.

3.2 Florist Implementation Design Style

The creation of tests in the Florist implementation should follow directly from the style guidelines listed in Section 3.1. There are a few files though that are implementation specific. Changes made to configuration files and the file `c-posix.c` should follow the style used within those files. Since those files are so large and only small modifications are needed there is no need to list an entire style guideline for them.

A separate makefile is used for the sockets portion of the Florist implementation. It should be of the following form:

```
include ../../Config

GNATMAKEFLAGS1 = -g -a -A../../floristlib -L../../floristlib
GNATMAKEFLAGS2= -cargs -gnatga -largs -lresolv -lflorist

TEST_FILES=\
 testfile1.adb\
 testfile2.adb\
 ...

TESTS=\
 testfile1\
 testfile2\
 ...

all: $(TESTS)

testfile1: testfile1.adb <testfile1 dependecies>
...

$(TESTS):  posix_report.ads posix_report.adb\
```

```
  ../../floristlib/libflorist.a test_parameters.ads test_parameters.adb
        gnatmake  $(GNATMAKEFLAGS1) $@ $(GNATMAKEFLAGS2)
clean:
        rm -f *.o *.ali $(TESTS)
```

Small additions can be made to the to the makefile but the general outline should be maintained.

4 Testing

Throughout the implementation process the tests were executed on the test platform. This platform is comprised of a quad HyperSparc system running Solaris 7. This system was chosen because it allows for testing of the parallel aspects of the POSIX.5b standard. Since some of the POSIX.5c tests for the Florist implementation depend upon operations in the POSIX.5b standard the same platform was used in POSIX.5c testing as in POSIX.5b testing. The operating system was chosen because it has a strong conformance to the POSIX.1g standard.

The tests were conducted in the order in which they were created. The first test file created was `p180400`. It contains only information that is localized to the `POSIX.Sockets` package, which corresponds to pieces of Section 18 of the POSIX.5c standard. Next the `pdd0100` test file was created. This package only uses code that can be found in the `POSIX.Sockets.Internet` package. The subprograms tested are part of Appendix D of the POSIX.5c standard.

Many of the tests depend upon subprograms found in the `POSIX.Sockets` and the `POSIX.Sockets.Internet` packages. To accommodate these tests file `p180401` was created. Finally, some tests require not only subprograms and constants from both `POSIX.socket` and `POSIX.Sockets.Internet` but also require multiple processes to correctly test certain subprograms. These tests form the `p180402` series of tests which include `p180402`, `p180402a`, `p180402b`, `p180402c`, `p180402d`, `p180402e`, `p180402f`, and `p180402g`. The information that provides the basis for these tests can be traced back to Section 18 and Appendix D of the POSIX.5c standard.

4.1 Package/Test Breakdown

The following table shows which test file contains which tests. Name represents the name of the constant, type, or subprogram. Class denotes if the object being tested is a constant, type, or subprogram. The errors field lists the the error conditions for which subprograms were tested, if any. Finally test file is where the test is located and parent package is where the the information needed for the test is located.

Name	Class	Errors	Test File	Parent Package
Socket_Type	type		p180400	POSIX.Sockets
Protocol_Family	type		p180400	POSIX.Sockets
Unspecified_Protocol	constant		p180400	POSIX.Sockets
Protocol_Number	type		p180400	POSIX.Sockets
Default_Protocol	constant		p180400	POSIX.Sockets
Socket_Address_Pointer	type		p180400	POSIX.Sockets
IO_Vector_Range	constant		p180400	POSIX.Sockets
Message_Option_Set	type		p180400	POSIX.Sockets
Message_Status_Set	type		p180400	POSIX.Sockets
Address_Flags	type		p180400	POSIX.Sockets
Set_Flags	subprogram		p180400	POSIX.Sockets
Get_Flags	subprogram		p180400	POSIX.Sockets
Set_Socket_Type	subprogram		p180400	POSIX.Sockets
Get_Socket_Type	subprogram		p180400	POSIX.Sockets
Get_Socket_Address_Info (3P)	subprogram	Name_Not_Known	p180400	POSIX.Sockets
Make_Empty	subprogram		p180400	POSIX.Sockets
Get_Socket_Error_Status	subprogram		p180400	POSIX.Sockets
Socket_Option_Value	type		p180400	POSIX.Sockets
Linger_Time	type		p180400	POSIX.Sockets
Connection_Queue_Length	type		p180400	POSIX.Sockets
Shutdown_Modes	type		p180400	POSIX.Sockets
Internet_Socket_Address_Pointer	type		pdd0100	POSIX.Sockets.Internet
Set_Internet_Port	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Internet_Port	subprogram		pdd0100	POSIX.Sockets.Internet
Is_Internet_Address	subprogram		pdd0100	POSIX.Sockets.Internet
String_To_Internet_Address	subprogram		pdd0100	POSIX.Sockets.Internet
Set_Internet_Address	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Internet_Address	subprogram		pdd0100	POSIX.Sockets.Internet
Is_Internet_Socket_Address	subprogram		pdd0100	POSIX.Sockets.Internet
Set_Socket_Name	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Socket_Name	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Address	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Network_Info_By_Name	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Network_Info_By_Address	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Name	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Family	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Network_Number	subprogram		pdd0100	POSIX.Sockets.Internet
For_Every_Network_Alias	subprogram		pdd0100	POSIX.Sockets.Internet
Open_Network_Database_Connection	subprogram		pdd0100	POSIX.Sockets.Internet
Close_Network_Database_Connection	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Protocol_Info_By_Name	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Protocol_Info_By_Number	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Protocol_Number	subprogram		pdd0100	POSIX.Sockets.Internet
For_Every_Protocol_Alias	subprogram		pdd0100	POSIX.Sockets.Internet
Open_Protocol_Database_Connection	subprogram		pdd0100	POSIX.Sockets.Internet
Close_Protocol_Database_Connection	subprogram		pdd0100	POSIX.Sockets.Internet
Keep_Alive_Time	constant		pdd0100	POSIX.Sockets.Internet
Socket_Retransmit_Time	constant		pdd0100	POSIX.Sockets.Internet
Wait_Forever	constant		pdd0100	POSIX.Sockets.Internet
Retransmit_Time_Default	constant		pdd0100	POSIX.Sockets.Internet
Set_Keep_Alive_Interval	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Keep_Alive_Interval	subprogram		pdd0100	POSIX.Sockets.Internet

Set_No_Delay	subprogram		pdd0100	POSIX.Sockets.Internet
Get_No_Delay	subprogram		pdd0100	POSIX.Sockets.Internet
Set_Retransmit_Time_Maximum	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Retransmit_Time_Maximum	subprogram		pdd0100	POSIX.Sockets.Internet
Set_Standardized_Urgent_Data	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Standardized_Urgent_Data	subprogram		pdd0100	POSIX.Sockets.Internet
Set_Segment_Size_Maximum	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Segment_Size_Maximum	subprogram		pdd0100	POSIX.Sockets.Internet
Time_To_Live	constant		pdd0100	POSIX.Sockets.Internet
Set_First_Hop	subprogram		pdd0100	POSIX.Sockets.Internet
Get_First_Hop	subprogram		pdd0100	POSIX.Sockets.Internet
Set_IP_Options	subprogram		pdd0100	POSIX.Sockets.Internet
Get_IP_Options	subprogram		pdd0100	POSIX.Sockets.Internet
Set_IP_Header_Options	subprogram		pdd0100	POSIX.Sockets.Internet
Get_IP_Header_Options	subprogram		pdd0100	POSIX.Sockets.Internet
IP_Header_Options_In_Use	subprogram		pdd0100	POSIX.Sockets.Internet
Reset_IP_Header_Options	subprogram		pdd0100	POSIX.Sockets.Internet
Set_Type_Of_Service	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Type_Of_Service	subprogram		pdd0100	POSIX.Sockets.Internet
Set_Initial_Time_To_Live	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Initial_Time_To_Live	subprogram		pdd0100	POSIX.Sockets.Internet
Set_Receive_Destination_Address	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Receive_Destination_Address	subprogram		pdd0100	POSIX.Sockets.Internet
Set_Type_Of_Service	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Type_Of_Service	subprogram		pdd0100	POSIX.Sockets.Internet
Get_Socket_Address_Info (4P)	subprogram	Service_Not_Supported Name_Not_Known Invalid_Flags Unknown_Protocol_...	p180401	POSIX.Sockets POSIX.Sockets.Internet
Create Set_Socket_Receive_Buffer_Size	subprogram subprogram	Protocol_Not_Supported Socket_Not_Supported Not_A_Socket No_Buffer_Space	p180401 p180401	POSIX.Sockets POSIX.Sockets.Internet POSIX.Sockets POSIX.Sockets.Internet
Get_Socket_Receive_Buffer_Size	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Set_Socket_Receive_Low_Water_Mark	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Get_Socket_Receive_Low_Water_Mark	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Set_Socket_Receive_Timeout	subprogram	Not_A_Socket Domain_Error	p180401	POSIX.Sockets POSIX.Sockets.Internet
Get_Socket_Receive_Timeout	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Set_Socket_Send_Buffer_Size	subprogram	Not_A_Socket No_Buffer_Space	p180401	POSIX.Sockets POSIX.Sockets.Internet
Get_Socket_Send_Buffer_Size	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Set_Socket_Send_Low_Water_Mark	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Get_Socket_Send_Low_Water_Mark	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Set_Socket_Send_Timeout	subprogram	Not_A_Socket Domain_Error	p180401	POSIX.Sockets POSIX.Sockets.Internet

Get_Socket_Send_Timeout	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Is_A_Socket	subprogram	Bad_File_Descriptor	p180401	POSIX.Sockets POSIX.Sockets.Internet
Set_Socket_Broadcast	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Get_Socket_Broadcast	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Set_Socket_Debugging	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Get_Socket_Debugging	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Set_Socket_Routing	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Get_Socket_Routing	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Set_Socket_Keep_Alive	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Get_Socket_Keep_Alive	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Set_Socket_Linger_Time	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Get_Socket_Linger_Time	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Set_Socket_OOB_Data_Inline	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Get_Socket_OOB_Data_Inline	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Set_Reuse_Address	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Get_Reuse_Address	subprogram	Not_A_Socket	p180401	POSIX.Sockets POSIX.Sockets.Internet
Bind	subprogram	Permission_Denied Already_In_Use Address_Not_Available Bad_File_Descriptor Not_A_Socket Inappropriate_Family Invalid_Argument	p180401	POSIX.Sockets POSIX.Sockets.Internet
Connect	subprogram	Is_Already_Connected Network_Unreachable Bad_File_Descriptor Not_A_Socket Inappropriate_Family Invalid_Argument Operation_In_Progress Connection_Refused Timed_Out	p180401	POSIX.Sockets POSIX.Sockets.Internet
Specify_Peer	subprogram	Bad_File_Descriptor Not_A_Socket Inappropriate_Family Invalid_Argument	p180401	POSIX.Sockets POSIX.Sockets.Internet
Unspecify_Peer	subprogram	Invalid_Argument	p180401	POSIX.Sockets POSIX.Sockets.Internet

Listen	subprogram	Bad_File_Descriptor Not_A_Socket Option_Not_Supported	p180401	POSIX.Sockets POSIX.Sockets.Internet
Shutdown	subprogram	Bad_File_Descriptor Not_A_Socket Not_Connected	p180401	POSIX.Sockets POSIX.Sockets.Internet
Accept_Connection	subprogram	Bad_File_Descriptor Not_A_Socket Invalid_Argument Option_Not_Supported Would_Block	p180401	POSIX.Sockets POSIX.Sockets.Internet
Receive (1)	subprogram	Bad_File_Descriptor Not_A_Socket Not_Connected Invalid_Argument Option_Not_Supported Would_Block	p180402	POSIX.Sockets POSIX.Sockets.Internet
Receive (2)	subprogram	Bad_File_Descriptor Not_A_Socket Not_Connected Invalid_Argument Option_Not_Supported Would_Block	p180402	POSIX.Sockets POSIX.Sockets.Internet
Receive (3)	subprogram	Bad_File_Descriptor Not_A_Socket Not_Connected Invalid_Argument Option_Not_Supported Would_Block	p180402	POSIX.Sockets POSIX.Sockets.Internet
Receive (4)	subprogram	Bad_File_Descriptor Not_A_Socket Not_Connected Invalid_Argument Option_Not_Supported Would_Block	p180402	POSIX.Sockets POSIX.Sockets.Internet
Receive_Message (1)	subprogram	Bad_File_Descriptor Not_A_Socket Not_Connected Option_Not_Supported Would_Block	p180402	POSIX.Sockets POSIX.Sockets.Internet
Receive_Message (2)	subprogram	Bad_File_Descriptor Not_A_Socket Not_Connected Option_Not_Supported Would_Block	p180402	POSIX.Sockets POSIX.Sockets.Internet
Send (1)	subprogram	Bad_File_Descriptor Not_A_Socket Not_Connected Invalid_Argument Option_Not_Supported Would_Block	p180402b	POSIX.Sockets POSIX.Sockets.Internet

Send (2)	subprogram	Bad_File_Descriptor Not_A_Socket Not_Connected Invalid_Argument Option_Not_Supported Would_Block	p180402e	POSIX.Sockets POSIX.Sockets.Internet
Send (3)	subprogram	Bad_File_Descriptor Not_A_Socket Not_Connected Invalid_Argument Option_Not_Supported Would_Block	p180402c	POSIX.Sockets POSIX.Sockets.Internet
Send (4)	subprogram	Bad_File_Descriptor Not_A_Socket Not_Connected Invalid_Argument Option_Not_Supported Would_Block	p180402g	POSIX.Sockets POSIX.Sockets.Internet
Send_Message (1)	subprogram	Bad_File_Descriptor Not_A_Socket Not_Connected Option_Not_Supported Would_Block	p180402d	POSIX.Sockets POSIX.Sockets.Internet
Send_Message (2)	subprogram	Bad_File_Descriptor Not_A_Socket Not_Connected Option_Not_Supported Would_Block	p180402f	POSIX.Sockets POSIX.Sockets.Internet

4.2 POSIX.5c Conformance Test Results

Because the POSIX.5c conformance tests were only conducted on one system and that system had been chosen for its conformance to the POSIX.1g only few errors were detected. The validity of all but one of the errors is still suspect. The one valid error is due to what may be a deficiency in the POSIX.5c standard. Unfortunately not all functions and error conditions were tested.

In a few situations tests were created and then removed. This was caused by what appeared to be a defect in the POSIX.5c standard. Results that were expected by the standard were simply inconsistent with what was occurring and the POSIX.1g standard was very vague in the area in question. This only occurred when dealing with error conditions. Unfortunately, the specification of what happens when an error occurs if very ambiguous in the POSIX.1g standard.

In addition to tests that were removed there are some tests that were never implemented. There are varied reasons for this. Some tests simply cannot be implemented because of the differences between Ada and C. Since Ada uses strong type checking, some errors that would occur in a C implementation will never make it past the compilation phase in Ada. Tests that required race conditions or system resources limits were the tests most commonly unimplemented. Virtually all of these sources were in tests that check for proper error codes when an exception occurs. Often setting up a race condition or system resource problem is very involved and requires large amounts of time to guarantee that the situation always occurs. The development of a sound test suite that was deliverable within a relatively narrow window of time was the top priority. Tests that require a long development time and are only used to test one or two error conditions were left out to meet the time constraints.

4.3 Florist Implementation Errors

When implementing the tests for the sockets portion of POSIX.5c several errors were found in the existing Florist implementation. This was expected as the sockets portion of the Florist implementation had never been thoroughly tested. The version of the Florist implementation that was used was not entirely up to date with the latest version of POSIX.5c. Therefore the code had to be brought up to date. Aside from that many small errors were found. Most of them were relatively simple. Appendix A contains a list of all the changes that were made to the `POSIX.Sockets` and `POSIX.Sockets.Internet` packages.

5 Future Additions to the Tests

Due to time and resource constraints it was not possible to follow every avenue of testing. There are many ways in which the above described tests could be extended to more thoroughly cover the conformance of a system with respect to the POSIX.5c standard. Several options are listed below.

1. With respect to the `Send` and `Receive` subprograms only one possible coupling has been used. It would be useful to create a situation in which all possible pairings of `Send` with `Receive` were used.
2. Some of the POSIX.5c packages have not had tests created for them. These include some of the lesser used socket packages (e.g. `POSIX.Sockets.Unix`) as well as all the XTI packages. By creating tests for the other socket packages new ways of testing the already tested socket packages would become available. Also there are a few subprograms in the `Posix.Sockets` package that could not be tested due to there reliance on one of the lesser used, untested socket packages.
3. Toy programs that simulate real world situations could be created to allow for less synthetic testing of all the packages. These toy programs could be a simple http server, telnet client, or any common program that is coded using sockets.
4. Once tests are written for the XTI packages comparisons could be made between the efficiency of socket operations and their XTI counterparts.
5. Toy programs could be created for the POSIX.1g subprograms that correspond to the toy programs created for the POSIX.5c subprograms. This would allow efficiency testing between the two standards.

6 Comments to Future Revisors

Creating tests for and maintaining the Florist implementation can be a daunting task. There are literary hundreds of files and over a hundred thousand lines of code. In addition to all the files one must maintain a copy of all significant changes made to any of the files. However, this task is not insurmountable. Listed in the following paragraphs are some comments that should make starting, finishing, and maintaining projects in the Florist implementation a little easier.

When starting work on the Florist implementation there are a couple files whose uses need to be immediately understood. However, it is not necessary to know what each line of code does before using them. The two files are the `Makefile` and `configure` files. Before doing any work or running any test `configure` should be executed. This will properly set up all the system depended information. The `configure` file only needs to be run once before starting and after a few special situations that are discussed later. This is good since it takes around 7 minutes to run the `configure` program. The other file, `Makefile` is used to compile all the package files into object code. Upon completion the `Makefile` moves all of the object code into the `floristlib` directory. The benefit of using a makefile is that only the pieces of code that have changed since the last compilation will be recompiled. This can be a great time saver as it takes about 10 minutes to compile every piece of the Florist implementation. Of course any listed times will vary based on the speed of the development platform.

The basic structure of the Florist implementation is the next piece of information that should be fully understood. All the packages that implement the functions described in the POSIX.5b and POSIX.5c standards are in Florist's root directory. From there all object code for those files is stored in the `floristlib` directory. Currently all POSIX.5b tests are located in the `tests` sub-directory. Any POSIX.5c tests are located in a subdirectory of the `tests` directory (e.g. `florist/tests/sockets` for the socket tests). All tests reference the object code stored in the `floristlib` directory.

Before writing any test the `tests/posix_report.ads` file should be thoroughly examined. It contains all the special functions that should be used when writing tests. While there uses may not be clear at first they are an invaluable and unavoidable piece of every test. Looking at some of the previously constructed tests may help to illuminate their uses. Virtually all of these subprograms have some form of guild lines set up in the Test Development Plan [3].

When writing conformance tests there are a few methodology decision that must be made. There is no single correct answer to these questions. However, once a question has been answered the same answer should be used whenever the same question arises. This should guarantee consistency in test development. The are two major questions that will appear, one is "What should be done with a test when the underlying system implementation does not conform to the POSIX.1g standard?". The two solutions are (1) don't implement the test and (2) implement the test even though testing it will give erroneous value. Both have their pros and cons. The other question is "The POSIX.1g and POSIX.5c standards differ, which should be used?". The two solutions are (1) implement the POSIX.1g standard even though there is no way to change the actual POSIX.5c to match. (2) Implement the POSIX.5c standard even though it is highly likely the statement in the POSIX.1g standard is what was really intended. Whichever solution is chosen it should be clearly documented and used throughout the entire series of tests.

Aside from creating and implementing tests, a record must be kept of all major changes made to any of the Florist files. This is done by using a modified version of the standard RCS (revision control system). Some of the commands are `checkout [-l]`, `checkin`, `forgetin`, and `writeablercs`. The major change over the original commands is with `checkin`. Not only does it check in a file but it removes the local copy and replaces it with a link to the master copy of all Florist files. Everyone working on the project checks into the same master files. If one person changes a file in the master files everyone will view this change since all files, except for ones being edited, are viewed from links to the master files. When `checkout -l` is used on a file it creates a local copy that can be edited. However, everyone else still views the old copy in the master files until the one that was checked out is checked back in. Then everyone automatically views the new file. Try to avoid using `checkout` without the -l option, as this creates a local copy that is not locked. It cannot be checked back in. Since links are only created when checking in a locked file, when using `checkout` there will soon be files that are only local and are not being edited. This means if the file is changed in the master files the local version will not reflect this, as only linked versions reflect the master files. If for some reason a file has been checked out using `checkout` just use `checkout -l` then `forgetin` to get the link back.

Finally, never forget that there is the distinct possibility that other people are working on the Florist files as well. It is very possible that changes made by one individual will affect many other's work. If things stop working, it might be a good idea to rerun `configure` and remake everything.

References

[1] *IEEE Standard for Information Technology – Portable Operating System Interface (POSIX) – Part xx: Protocol Independent Interfaces (PII)* (IEEE Std 1003.1g), 1996

[2] *IEEE Standard for Information Technology – POSIX Ada Interfaces – Part 1: Binding for System Application Program Interface (API)* (IEEE Std 1005.1c-1998). 1998

[3] Baker, T.P. *POSIX 1003.5b Test Suite: Milestone 4 Test Development Plan*, 1998

A Changes Made in Package Files

Difference in posix-sockets.gps

```
62a63
>    Null_Socket_Address : constant Socket_Address_Pointer;
77c78
<        Pointer : in     IO_Vector_Array_Pointer);
---
>        Ptr     : in     IO_Vector_Array_Pointer);
103c104
<        Length  : in     Positive);
---
>        Length  : in     POSIX.IO_Count);
143c144
<    type Socket_Address_Info is private;
---
>    type Socket_Address_Info is limited private;
307c308
<        From             : out Socket_Address_Pointer;
---
>        From             : in  Socket_Address_Pointer;
316c317
<        From             : out Socket_Address_Pointer;
---
>        From             : in  Socket_Address_Pointer;
392c393
<    type Socket_Address (Length : POSIX.C.size_t) is
---
>    type Socket_Address is
394c395,396
<       C : aliased POSIX.C.Sockets.struct_sockaddr;
---
>       Length : POSIX.C.size_t := 0;
>       C      : aliased POSIX.C.Sockets.struct_sockaddr;
397a400
>    Null_Socket_Address : constant Socket_Address_Pointer := null;
```

Difference in posix-sockets.gpb

```
40,42c40
< with Ada.Streams,
<      Ada.IO_Exceptions,
<      POSIX,
---
> with POSIX,
47a46
>
76,77d74
<      (char_ptr, addrinfo_ptr);
<    function To_addrinfo_ptr is new Unchecked_Conversion
118,160d114
<    --  Internet support from <netdb.h>
<    --  Note : using the thread safe versions of the following rather
<    --         than the MT unsafe versions spec'd in dot1g
< --     function c_getnetbyname_r
< --        (name : char_ptr; result : netent_ptr; buffer : char_ptr;
< --         buflen : int)
< --       return netent_ptr;
< --     pragma Import (C, c_getnetbyname_r, getnetbyname_r_LINKNAME);
<
< --     function c_getnetbyaddr_r
< --        (net : unsigned; typ : int; result : netent_ptr;
< --         buffer : char_ptr; buflen : int)
< --       return netent_ptr;
< --     pragma Import (C, c_getnetbyaddr_r, getnetbyaddr_r_LINKNAME);
<
< --     function c_setnetent (stayopen : int) return int;
< --     pragma Import (C, c_setnetent, setnetent_LINKNAME);
<
< --     function c_endnetent return int;
< --     pragma Import (C, c_endnetent, endnetent_LINKNAME);
<
< --     function c_getprotobyname_r
< --        (name : char_ptr; result : protoent_ptr; buffer : char_ptr;
< --         buflen : int)
< --       return protoent_ptr;
< --     pragma Import (C, c_getprotobyname_r, getprotobyname_r_LINKNAME);
<
< --     function c_getprotobynumber_r
< --        (proto : int; result : protoent_ptr; buffer : char_ptr; buflen : int)
< --       return protoent_ptr;
< --     pragma Import (C, c_getprotobynumber_r, getprotobynumber_r_LINKNAME);
<
< --     function c_setprotoent (stayopen : int) return int;
< --     pragma Import (C, c_setprotoent, setprotoent_LINKNAME);
<
< --     function c_endprotoent return int;
< --     pragma Import (C, c_endprotoent, endprotoent_LINKNAME);
<
<    --  poll file descriptors from <sys/poll.h>
< --     function c_poll (fds : pollfd_ptr; nfds : unsigned; timeout : int)
< --        return int;
< --     pragma Import (C, c_poll, poll_LINKNAME);
<
178,185d131
<    --  select file descriptors from <sys/select.h>
<
< --     function c_select
< --        (nfds : int; readfds : fd_set_ptr; writefds : fd_set_ptr;
< --         exceptfds : fd_set_ptr; timeout : timeval_ptr)
< --        return int;
< --     pragma Import (C, c_select, select_LINKNAME);
<
204c150
<       Message.C.msg_name := To_char_ptr (Name.C'Address);
---
>       Message.C.msg_name := Name.C'Unchecked_Access;
210c156
<        Pointer : in     IO_Vector_Array_Pointer) is
---
>        Ptr     : in     IO_Vector_Array_Pointer) is
212,214c158,160
<       Message.C.msg_iov := To_iovec_ptr (Pointer (Pointer'First)'Address);
<       Message.C.msg_iovlen := size_t (Pointer.all'Length);
<       Message.io_vector_array_ptr := Pointer;
---
>       Message.C.msg_iov := To_iovec_ptr (Ptr (Ptr'First)'Address);
>       Message.C.msg_iovlen := size_t (Ptr.all'Length);
>       Message.io_vector_array_ptr := Ptr;
254c200
<       Length  : in     Positive) is
---
>       Length  : in     POSIX.IO_Count) is
274a221
>          Length := 0;
295c242
<       Name              : out Socket_Address_Pointer)
---
>       Name              : in  Socket_Address_Pointer)
428a376,378
>    procedure c_freeaddrinfo (ai : addrinfo_ptr);
>    pragma Import (C, c_freeaddrinfo, freeaddrinfo_LINKNAME);
>
432c382,383
<       null;
---
>       null;  -- remove later
>       c_freeaddrinfo (Info_Item.C'Unchecked_Access);
449c400
<          (Option_Set'(Option => To_Bits (Info_Item.C.ai_family)));
---
>          (Option_Set'(Option => To_Bits (Info_Item.C.ai_flags)));
470a422
>
489,492d440
< --   function Get_Address (Info_Item: Socket_Address_Info)
< --     return Socket_Address_Pointer is abstract
< --  this function is protocol specific. Look in the sockets child packages.
<
515a464,468
>       --  Not 100 percent why but you need this object to be local or
>       --  else the function can cause problems.  It may be that its being
>       --  passed in by registers (in parameter) and the pointers don't
>       --  work.
>       Tmp_Request : Socket_Address_Info := Request;
525c478
<                                Request.C'Unchecked_Access,
---
>                                Tmp_Request.C'Unchecked_Access,
536c489,490
<          Check (Result);
---
>          Result := int (Get_Error_Code);
>          Check_NZ (Result);
573c527,528
<          Check (Result);
---
>          Result := int (Get_Error_Code);
>          Check_NZ (Result);
586c541
<          next_item := To_addrinfo_ptr (List.C.ai_next);
---
>          next_item := List.C.ai_next;
590c545
<             next_item := To_addrinfo_ptr (next_item.all.ai_next);
---
>             next_item := next_item.all.ai_next;
1142c1097
<       Restore_Signals (Old_Mask'Unchecked_Access);
---
>       Restore_Signals (Masked_Signals, Old_Mask'Unchecked_Access);
1206c1161
<        From             : out Socket_Address_Pointer;
---
>        From             : in  Socket_Address_Pointer;
1212c1167
<       Address_Length : aliased size_t;
---
>       Address_Length : aliased size_t := 0;
1220c1175,1177
<       Address_Length := From.Length;
---
>       if From /= Null_Socket_Address then
>          Address_Length := From.Length;
>       end if;
1227c1184
<       Restore_Signals (Old_Mask'Unchecked_Access);
```

```
---
>       Restore_Signals (Masked_Signals, Old_Mask'Unchecked_Access);
1246c1203
<        From             : out Socket_Address_Pointer;
---
>        From             : in  Socket_Address_Pointer;
1257c1214,1216
<       Address_Length := From.Length;
---
>       if From /= Null_Socket_Address then
>          Address_Length := From.Length;
>       end if;
1300c1259
<       Restore_Signals (Old_Mask'Unchecked_Access);
---
>       Restore_Signals (Masked_Signals, Old_Mask'Unchecked_Access);
1360c1319
<       Restore_Signals (Old_Mask'Unchecked_Access);
---
>       Restore_Signals (Masked_Signals, Old_Mask'Unchecked_Access);
1428c1387
<       Restore_Signals (Old_Mask'Unchecked_Access);
---
>       Restore_Signals (Masked_Signals, Old_Mask'Unchecked_Access);
1487c1446
<       Restore_Signals (Old_Mask'Unchecked_Access);
---
>       Restore_Signals (Masked_Signals, Old_Mask'Unchecked_Access);
```

Difference in posix-sockets-internet.ads

```
75c75
<        Value : in     Internet_Port);
---
>        Port  : in     Internet_Port);
105c105
<    type Network_Number is range 0 .. 65535;
---
>    type Network_Number is range 0 .. (2**32 -1);
237a238
>    type IP_Ancillary_Data_Pointer is access all IP_Ancillary_Data;
240,242c241,243
<        Data    : in     IP_Ancillary_Data);
<    function Get_Ancillary_Data
<       (Message : Socket_Message)
---
>        Data    : in     IP_Ancillary_Data_Pointer);
>    function Get_Destination_Address
>       (Data : IP_Ancillary_Data)
253a255,256
>       Length : POSIX.C.size_t := POSIX.C.Sockets.struct_sockaddr_in'Size /
>                                  System.Storage_Unit;
```

Difference in posix-sockets-internet.gpb

```
41d40
< with GNAT.IO; use GNAT.IO;
45c44,45
< with Text_IO;
---
> --   with Ada.Text_IO; use Ada.Text_IO;
> --   with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
61,63d60
<    function To_ADA is new Unchecked_Conversion
<       (char_ptr, IP_Ancillary_Data_Access);
<
70,75d66
<    function To_sockaddr_in_var_ptr is new Unchecked_Conversion
<       (char_ptr, sockaddr_in_var_ptr);
<
<    function To_sockaddr_in_ptr is new Unchecked_Conversion
<       (char_ptr, sockaddr_in_ptr);
<
119c110
<        Value : in     Internet_Port) is
---
>        Port  : in     Internet_Port) is
121c112
<       Name.C.sin_port := in_port_t (Value);
---
>       Name.C.sin_port := in_port_t (Port);
184a176,181
>
>    function strlen (str : in char_ptr) return size_t;
>    pragma Import (C, strlen, "strlen");
>
>
>
189c186
<            (c_inet_ntoa (Address.C.s_addr'Address));
---
>          (c_inet_ntoa (Address.C.s_addr'Address));
226,227c223,224
<       (net : unsigned;
<        typ : int;
---
>       (net    : unsigned;
>        typ    : int;
235c232
<       (Number : Network_Number;
---
>       (Number  : Network_Number;
240a238
>       Temp   : Error_Code;
241a240,245
>       --  Since the error is only caused under special conditions
>       --  we have to make sure an old one is not left in the error
>       --  number.
>       if Fetch_Errno = ERANGE then
>          Store_Errno (No_Error);
>       end if;
244c248
<        To_char_ptr (Storage (Storage'First)'Address), Storage'Length);
---
>         To_char_ptr (Storage (Storage'First)'Address), Storage'Length);
248,251c252,259
<          return (C => (n_name     => null,
<                        n_aliases  => null,
<                        n_addrtype => 0,
<                        n_net      => 0));
---
>          Temp := Fetch_Errno;
>          if Temp = ERANGE then
>             raise Constraint_Error;
>          else return (C => (n_name     => null,
>                       n_aliases  => null,
>                       n_addrtype => 0,
>                       n_net      => 0));
>          end if;
256c264
<       (name : char_ptr;
---
>       (name   : char_ptr;
267,269c275,285
<       netent : aliased POSIX.C.NetDB.struct_netent;
<       Result : POSIX.C.NetDB.netent_ptr;
<    begin
---
>       netent   : aliased POSIX.C.NetDB.struct_netent;
>       Result   : POSIX.C.NetDB.netent_ptr;
>       Name_NUL : POSIX_String := Name & NUL;
>       Temp     : POSIX.Error_Code;
>    begin
>       --  Since the error is only caused under special conditions
>       --  we have to make sure an old one is not left in the error
>       --  number.
>       if Fetch_Errno = ERANGE then
>          Store_Errno (No_Error);
>       end if;
271c287
<       (Name (Name'First)'Unchecked_Access, netent'Unchecked_Access,
---
>       (Name_NUL (Name_NUL'First)'Unchecked_Access, netent'Unchecked_Access,
276,279c292,300
<          return (C => (n_name => null,
<                        n_aliases => null,
<                        n_addrtype => 0,
<                        n_net => 0));
---
>          Temp := Fetch_Errno;
>          if Temp = ERANGE then
>             raise Constraint_Error;
>          else
>             return (C => (n_name => null,
>                           n_aliases => null,
>                           n_addrtype => 0,
>                           n_net => 0));
>          end if;
340c361
<       (Number : Protocol_Number;
---
>       (Number  : Protocol_Number;
344c365,366
<       Result : protoent_ptr;
---
>       Result   : protoent_ptr;
>       Temp     : POSIX.Error_Code;
345a368,373
>       --  Since the error is only caused under special conditions
>       --  we have to make sure an old one is not left in the error
>       --  number.
>       if Fetch_Errno = ERANGE then
>          Store_Errno (No_Error);
>       end if;
352,354c380,387
<          return (C => (p_name => null,
<                        p_aliases => null,
<                        p_proto => 0));
---
>          Temp := Fetch_Errno;
>          if Temp = ERANGE then
>             raise Constraint_Error;
>          else
>             return (C => (p_name => null,
>                           p_aliases => null,
>                           p_proto => 0));
>          end if;
367c400
<       (Name : POSIX.POSIX_String;
---
```

```
>       (Name    : POSIX.POSIX_String;
371,372c404,413
<       Result : protoent_ptr;
<    begin
---
>       Result   : protoent_ptr;
>       Name_NUL : POSIX.POSIX_String := Name & NUL;
>       Temp     : POSIX.Error_Code;
>    begin
>       --  Since the error is only caused under special conditions
>       --  we have to make sure an old one is not left in the error
>       --  number.
>       if Fetch_Errno = ERANGE then
>          Store_Errno (No_Error);
>       end if;
374c415,416
<       (Name (Name'First)'Unchecked_Access, protoent'Unchecked_Access,
---
>       (Name_NUL (Name_NUL'First)'Unchecked_Access,
>        protoent'Unchecked_Access,
379,381c421,428
<          return (C => (p_name => null,
<                        p_aliases => null,
<                        p_proto => 0));
---
>          Temp := Fetch_Errno;
>          if Temp = ERANGE then
>             raise Constraint_Error;
>          else
>             return (C => (p_name => null,
>                           p_aliases => null,
>                           p_proto => 0));
>          end if;
666c713
<       Check (c_getsockopt (int (Socket), int (IPPROTO_UDP),
---
>       Check (c_getsockopt (int (Socket), int (IPPROTO_IP),
683c730
<       Check (c_setsockopt (int (Socket), int (IPPROTO_UDP),
---
>       Check (c_setsockopt (int (Socket), int (IPPROTO_IP),
689c736
<        Data    : in     IP_Ancillary_Data) is
---
>        Data    : in     IP_Ancillary_Data_Pointer) is
699,700c746,747
<    function Get_Ancillary_Data
<       (Message : Socket_Message)
---
>    function Get_Destination_Address
>       (Data : IP_Ancillary_Data)
706d752
<       Data : IP_Ancillary_Data_Access := To_ADA (Message.C.msg_control);
708c754
<       return (C => Data.all.C2);
---
>       return (C => Data.C2);
710c756
<    end Get_Ancillary_Data;
---
>    end Get_Destination_Address;
761c807,808
<       return (C => c_address);
---
>       return (C      => c_address,
>               Length => c_address_len);
785c832,833
<       return (C => c_address);
---
>       return (C      => c_address,
>               Length => c_address_len);
798c846,847
<       return (C => To_sockaddr_in_ptr (Handle.C.msg_name).all);
---
>       return (C => To_sockaddr_in_ptr (Handle.C.msg_name).all,
>               Length => Handle.C.msg_namelen);
810c859,860
<       return (C => To_sockaddr_in_ptr (Info_Item.C.ai_addr).all);
---
>       return (C => To_sockaddr_in_ptr (Info_Item.C.ai_addr).all,
>               Length => Info_Item.C.ai_addrlen);
826c876,878
<       if (Ptr.C.sa_family = POSIX.C.Sockets.AF_INET) then
---
>       if Ptr = null then
>          return null;
>       elsif (Ptr.C.sa_family = POSIX.C.Sockets.AF_INET) then
838c890,892
<       if (Ptr.C.sa_family = POSIX.C.Sockets.AF_INET) then
---
>       if Ptr = null then
>          return false;
>       elsif (Ptr.C.sa_family = POSIX.C.Sockets.AF_INET) then
```

B Code For Tests

p180400.adb

```
-----------------------------------------------------------------------------
--                                                                         --
--                      POSIX.5c VALIDATION TEST SUITE                     --
--                                                                         --
--                             P 1 8 0 4 0 0                               --
--                                                                         --
--                                B o d y                                  --
--                                                                         --
--                                                                         --
--  Copyright (c) 1995-1999 Florida  State  University  (FSU).  All Rights --
--  Reserved.                                                              --
--                                                                         --
--  This is free software;  you can redistribute it and/or modify it under --
--  terms of the  GNU  General  Public  License  as published by the  Free --
--  Software Foundation;  either version 2, or (at your option) any  later --
--  version.  This  software  is distributed  in the hope that it  will be --
--  useful, but WITHOUT ANY WARRANTY; without even the implied warranty of --
--  MERCHANTABILITY   or  FITNESS FOR A PARTICULAR PURPOSE.   See the  GNU --
--  General Public License for more details.  You  should have  received a --
--  copy of the GNU General Public License  distributed  with  GNARL;  see --
--  file  COPYING.  If not,  write to  the  Free  Software  Foundation, 59 --
--  Temple Place - Suite 330, Boston, MA 02111-1307, USA.                  --
--                                                                         --
--  Under contract  GS-35F-4506G, the U. S. Government obtained  unlimited --
--  rights in the software and documentation contained herein.   Unlimited --
--  rights are defined in DFAR 252,227-7013(a)(19).  By making this public --
--  release,   the  Government  intends  to  confer  upon  all  recipients --
--  unlimited  rights equal to those held by the Government.  These rights --
--  include rights to use,  duplicate,  release  or  disclose the released --
--  data an computer software  in whole or in part,  in any manner and for --
--  any purpose whatsoever, and to have or permit others to do so.         --
--                                                                         --
--  DISCLAIMER   --   ALL MATERIALS OR INFORMATION HEREIN RELEASED,   MADE --
--  AVAILABLE OR DISCLOSED ARE AS IS.   THE GOVERNMENT MAKES NO EXPRESS OR --
--  IMPLIED WARRANTY AS TO ANY MATTER WHATSOEVER, INCLUDING THE CONDITIONS --
--  OF THE SOFTWARE,  DOCUMENTATION  OR  OTHER INFORMATION RELEASED,  MADE --
--  AVAILABLE OR DISCLOSED,  OR THE OWNERSHIP,  MERCHANTABILITY, OR FITNESS --
--  FOR A PARTICULAR PURPOSE OF SAID MATERIAL.                             --
--                                                                         --
-----------------------------------------------------------------------------

--  Basic test for package POSIX_Sockets,
--  in IEEE Std 1003.5c Section 18.4.

--  This test covers only features that depend only on
--  the package itself and features from other packages
--  that are required to be supported.

with POSIX,
     POSIX_Sockets,
     POSIX_Report,
     POSIX_IO,
     Test_Parameters;

procedure p180400 is

   use POSIX,
       POSIX_Sockets,
       POSIX_IO,
       POSIX_Report,
       Test_Parameters;

   Unknown_Name    : POSIX_String := "xi.csfsu.edu";
   Unknown_Service : POSIX_String := "retelnet";
   Valid_Service   : POSIX_String := "telnet";

   procedure Get_Socket_Address_Info_3Param_Tests
      (Name     : in POSIX_String    := Valid_Internet_Name;
       Service  : in POSIX_String    := Valid_Service;
       Expected : in Error_Code      := No_Error;
       Error1   : in String          := "";
       Error2   : in String;
       Error3   : in String);

   ------------------------------------------------------------------------
   --  Procedure used to test Create_Pair method

   procedure Get_Socket_Address_Info_3Param_Tests
      (Name     : in POSIX_String    := Valid_Internet_Name;
       Service  : in POSIX_String    := Valid_Service;
       Expected : in Error_Code      := No_Error;
       Error1   : in String          := "";
       Error2   : in String;
       Error3   : in String)
   is
      Info : Socket_Address_Info_List;
   begin
      Get_Socket_Address_Info (Name, Service, Info);
      if Error1 /= "" then
         Expect_Exception (Error1);
      end if;
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Expected then
         Optional (Network_Management_Option,
            Unknown_Protocol_Option, E1, Error2);
      end if;
   when E2 : others => Unexpected_Exception (E2, Error3);
   end Get_Socket_Address_Info_3Param_Tests;

-----------------------------------------------------------------------
--  Begin Tests

begin
   Header ("p180400");
   Test ("package POSIX.Sockets");

   --------------------------------------------------------------------
   --  Socket Types are in the proper positions

   Test ("Socket Types Position [18.4.1.1]");
   declare
      Type_of_Socket : Socket_Type;
   begin
      Type_of_Socket := 0;
      Assert (Socket_Type'First = 0 and
              Socket_Type'Pos (Datagram_Socket) = 1 and
              Socket_Type'Pos (Stream_Socket) = 2 and
              Socket_Type'Pos (Raw_Socket) = 4 and
              Socket_Type'Last = 6, "A001");
   exception when E : others => Unexpected_Exception (E, "A002");
   end;

   --------------------------------------------------------------------
   --  Socket Types are unique

   Test ("Socket Types Uniqueness [18.4.1.1]");
   declare
      A : constant array (1 .. 5) of Socket_Type :=
        (Stream_Socket,
         Datagram_Socket,
         Raw_Socket,
         Sequenced_Packet_Socket,
         Unspecified_Socket_Type);
   begin
      for I in A'Range loop
         for J in A'Range loop
            Assert (A (I) /= A (J) or I = J, "A003");
         end loop;
      end loop;
   exception when E : others => Unexpected_Exception (E, "A004");
   end;

   --------------------------------------------------------------------
   --  Protocol Family has the appropriate range

   Test ("Protocol Family  [18.4.1.1]");
   declare
      Proto_Family   : Protocol_Family;
   begin
      Proto_Family := 0;
      Assert (Protocol_Family'First = 0 and
              Protocol_Family'Last = 25, "A005");
   exception when E : others => Unexpected_Exception (E, "A006");
   end;

   --------------------------------------------------------------------
   --  Unspecified Protocol has been declared with the proper value

   Test ("Unspecified Protocol  [18.4.1.1]");
   begin
      Assert (Unspecified_Protocol = 0, "A007");
   exception when E : others => Unexpected_Exception (E, "A008");
   end;

   --------------------------------------------------------------------
   --  Protocol_Number has the appropriate range

   Test ("Protocol Number  [18.4.1.1]");
   declare
      Proto_Number   : Protocol_Number;
   begin
      Proto_Number := 0;
      Assert (Protocol_Number'First = 0 and
              Protocol_Number'Last = 65535, "A009");
   exception when E : others => Unexpected_Exception (E, "A010");
   end;

   --------------------------------------------------------------------
   --  Default_Protocol has been declared with the proper value

   Test ("Default Protocol  [18.4.1.1]");
   begin
      Assert (Default_Protocol = 0, "A011");
   exception when E : others => Unexpected_Exception (E, "A012");
   end;

   --------------------------------------------------------------------
   --  Socket_Address_Pointer type has been propely declared

   Test ("Socket Addresses  [18.4.1.2]");
   declare
      SocAddrPtr : Socket_Address_Pointer;
   begin
      SocAddrPtr := Null_Socket_Address;
   exception when E : others => Unexpected_Exception (E, "A013");
   end;

   --------------------------------------------------------------------
   --  IO_Vector_Range has the proper range
```

```
Test ("Socket Messages : IO_Vector_Range  [18.4.1.3]");
declare
   IOVRange : IO_Vector_Range;
begin
   IOVRange := 12;
   Assert (IO_Vector_Range'First = 1, "A014");
   Assert (IO_Vector_Range'Last = Natural'Last, "A015");
exception when E : others => Unexpected_Exception (E, "A016");
end;

-----------------------------------------------------------------------
--  All options in Message_Option_Set are unique

Test ("Message_Option_Set [18.4.1.3]");
declare
   A : constant array (1 .. 4) of Message_Option_Set :=
      (Peek_Only,
       Process_OOB_Data,
       Wait_For_All_Data,
       Do_Not_Route);
begin
   for I in A'Range loop
      for J in A'Range loop
         Assert (A (I) /= A (J) or I = J, "A017");
      end loop;
   end loop;
exception when E : others => Unexpected_Exception (E, "A018");
end;

-----------------------------------------------------------------------
--  All Message_Status_Set options are unique

Test ("Message_Status_Set [18.4.1.3]");
declare
   A : constant array (1 .. 4) of Message_Status_Set :=
      (Received_OOB_Data,
       End_Of_Message,
       Message_Truncated,
       Ancillary_Data_Lost);
begin
   for I in A'Range loop
      for J in A'Range loop
         Assert (A (I) /= A (J) or I = J, "A019");
      end loop;
   end loop;
exception when E : others => Unexpected_Exception (E, "A020");
end;

-------------------------------------------------------------------------
--  Create_Pair [18.4.6] cannot be tested until sockets-unix has been  --
--  implemented.                                                       --
--                                                                     --
-------------------------------------------------------------------------

-----------------------------------------------------------------------
--  The Address_Flags options are all unique

Test ("Address_Flags type [18.4.7]");
declare
   A : constant array (1 .. 2) of Address_Flags :=
     (Use_For_Binding,
      Canonical_Name);
begin
   for I in A'Range loop
      for J in A'Range loop
         Assert (A (I) /= A (J) or I = J, "A055");
      end loop;
   end loop;
exception when E : others => Unexpected_Exception (E, "A056");
end;

-----------------------------------------------------------------------
--  If the Network Management option is supported then the get and set
--  methods are consistent

Test ("Set/Get_Flags Test [18.4.7]");
declare
   Info_Item : Socket_Address_Info;
   Flags     : Address_Flags;
begin
   Flags := Canonical_Name;
   Set_Flags (Info_Item, Flags);
   Flags := Use_For_Binding;

   --  The Assert statement is used to garauntee that the optimizer
   --  doesn't remove the previous assignment due to it being dead code.
   Assert (Flags = Use_For_Binding, "A057");
   Flags := Get_Flags (Info_Item);
   Assert (Flags = Canonical_Name, "A058");
exception
when E1 : POSIX_Error =>
   Optional (Network_Management_Option,
      Unknown_Protocol_Option, E1, "A059");
when E2 : others => Unexpected_Exception (E2, "A060");
end;

-----------------------------------------------------------------------
--  If the Network Management option is supported then the Set and Get
--  methods are consistent

Test ("Set/Get_Socket_Type Test [18.4.7]");
declare
   Info_Item : Socket_Address_Info;
   To        : Socket_Type;
begin
   To := Stream_Socket;
   Set_Socket_Type (Info_Item, To);
   To := Raw_Socket;

   --  The Assert statement is used to garauntee that the optimizer
   --  doesn't remove the previous assignment due to it being dead code.
   Assert (To = Raw_Socket, "A065");
   To := Get_Socket_Type (Info_Item);
   Assert (To = Stream_Socket, "A066");
exception
when E1 : POSIX_Error =>
   Optional (Network_Management_Option,
      Unknown_Protocol_Option, E1, "A067");
when E2 : others => Unexpected_Exception (E2, "A068");
end;

-----------------------------------------------------------------------
--  If the Network Management option is supported then Info is filled
--  up with the appropriate information

Test ("Get_Socket_Address_Info (3-param version) [18.4.7]");
begin
   Get_Socket_Address_Info_3Param_Tests (Error2 => "A073",
      Error3 => "A074");
end;

-----------------------------------------------------------------------
--  If the Network Management option is supported then Info is filled
--  up with the appropriate information

Test ("Get_Socket_Address_Info (3-param with null Name) [18.4.7]");
begin
   Get_Socket_Address_Info_3Param_Tests (Name => "",
      Error2 => "A075", Error3 => "A076");
end;

-----------------------------------------------------------------------
--  If the Network Management option is supported then Info is filled
--  up with the appropriate information

Test ("Get_Socket_Address_Info (3-param with null Service) [18.4.7]");
begin
   Get_Socket_Address_Info_3Param_Tests (Service => "",
      Error2 => "A077", Error3 => "A078");
end;

-----------------------------------------------------------------------
--  If the Network Management option is supported then the
--  Name_Not_Known POSIX_Error code will be raised

Test
  ("Get_Socket_Address_Info (3-param): Name_Not_Known (Name) [18.4.7.3]");
begin
   Get_Socket_Address_Info_3Param_Tests (Name => Unknown_Name,
      Expected => Name_Not_Known, Error1 => "A079",
      Error2 => "A080", Error3 => "A081");
end;

-----------------------------------------------------------------------
--  If the Network Management option is supported then the
--  Name_Not_Known POSIX_Error code will be raised

Test
("Get_Socket_Address_Info (3-param): Name_Not_Known (Service) [18.4.7.3]");
begin
   Get_Socket_Address_Info_3Param_Tests (Service => Unknown_Service,
      Expected => Name_Not_Known, Error1 => "A082",
      Error2 => "A083", Error3 => "A084");
end;

-----------------------------------------------------------------------
--  If the Network Management option is supported then the
--  Name_Not_Known POSIX_Error code will be raised

Test
  ("Get_Socket_Address_Info (3-param): Name_Not_Known (Null) [18.4.7.3]");
begin
   Get_Socket_Address_Info_3Param_Tests (Name => "", Service => "",
      Expected => Name_Not_Known, Error1 => "A085",
      Error2 => "A086", Error3 => "A087");
end;

-----------------------------------------------------------------------
--  If the Network Management option is supported then the Info object
--  will be emptied

Test ("Make_Empty [18.4.7]");
declare
   Info : Socket_Address_Info_List;
begin
   Get_Socket_Address_Info (Valid_Internet_Name, Valid_Service, Info);
   Make_Empty (Info);
exception
when E1 : POSIX_Error =>
   Optional (Network_Management_Option,
      Operation_Not_Implemented, E1, "A088");
when E2 : others => Unexpected_Exception (E2, "A089");
end;

-----------------------------------------------------------------------
--  If the Network Management option is supported then the
--  Unknown_Socket_type POSIX_Error code will be raised

--  This test is not working at the moment.  The error occurs in
--  the getaddrinfo.c file under a bad socket type.  The problem
```

```
   --  occurs when dereferencing the ai_addr pointer.  Not for sure
   --  what the exact cause is.  Raises STRORAGE Errror.

   ------------------------------------------------------------------------
   --  Since the socket was never created Get_Socket_Error_Status will
   --  return Not_A_Socket

   Test ("Not_A_Socket  [18.4.8.3]");
   declare
      Sock  : POSIX_IO.File_Descriptor := 0;
      Error : POSIX.Error_Code := 0;
   begin
      Error := Get_Socket_Error_Status (Sock);
      Expect_Exception ("A115");
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Not_A_Socket then
         Unexpected_Exception (E1, "A116");
      end if;
   when E2 : others => Unexpected_Exception (E2, "A117");
   end;

   ------------------------------------------------------------------------
   --  Socket_Option_Values are set and in the proper order

   Test ("Socket_Option_Values [18.4.9]");
   declare
      Option_Value : Socket_Option_Value := Enabled;
   begin
      Option_Value := Disabled;
      Assert (Socket_Option_Value'First = Enabled and
         Socket_Option_Value'Last = Disabled, "A124");
   exception when E : others => Unexpected_Exception (E, "A125");
   end;

   ------------------------------------------------------------------------
   --  Subtype Linger_Time has the appropriate bounds


   Test ("Linger_Time subtype bounds [18.4.9]");
   declare
      L_Time : Linger_Time;
   begin
      L_Time := 2;
      Assert (Linger_Time'First = 0, "A174");
      Assert (Linger_Time'Last = POSIX.Seconds'Last, "A175");
   exception when E : others => Unexpected_Exception (E, "A176");
   end;

   ------------------------------------------------------------------------
   --  Connection_Queue_Length/_Maximum have the appropriate values

   Test ("Connection_Queue_Length/_Maximum [18.4.11]");
   declare
      Queue_Length : Connection_Queue_Length;
   begin
      Queue_Length := 3;
      Assert (Connection_Queue_Length_Maximum = 5, "A309");
      Assert (Connection_Queue_Length'First = 0, "A310");
      Assert (Connection_Queue_Length'Last = 5, "A311");
   exception when E : others => Unexpected_Exception (E, "A312");
   end;

   ------------------------------------------------------------------------
   --  Shutdown_Mode has the appropriate values

   Test ("Shutdown_Mode [18.4.11]");
   declare
      Mode : Shutdown_Mode;
   begin
      Mode := Further_Receives_Disallowed;
      Assert (Further_Receives_Disallowed = Shutdown_Mode'First and
         Shutdown_Mode'Pos (Further_Sends_Disallowed) = 1 and
         Further_Sends_And_Receives_Disallowed = Shutdown_Mode'Last,
         "A313");
   exception when E : others => Unexpected_Exception (E, "A314");
   end;

   ------------------------------------------------------------------------
   --  The other functions are protocol dependent and will be tested with
   --  the appropriate child package
   ------------------------------------------------------------------------

   Done;
exception when E : others => Fatal_Exception (E, "A999");
end p180400;
```

pdd0100.adb

```
------------------------------------------------------------------------------
--                                                                          --
--                      POSIX.5c VALIDATION TEST SUITE                      --
--                                                                          --
--                             P D D 0 1 0 0                                --
--                                                                          --
--                                 B o d y                                  --
--                                                                          --
--                                                                          --
--  Copyright (c) 1995-1999 Florida  State  University  (FSU).  All Rights  --
--  Reserved.                                                               --
--                                                                          --
--  This is free software;  you can redistribute it and/or modify it under  --
--  terms of the  GNU  General  Public  License  as published by the  Free  --
--  Software Foundation;  either version 2, or (at your option) any  later  --
--  version.  This  software  is distributed  in the hope that it  will be  --
--  useful, but WITHOUT ANY WARRANTY; without even the implied warranty of  --
--  MERCHANTABILITY   or  FITNESS FOR A PARTICULAR PURPOSE.   See the  GNU  --
--  General Public License for more details.  You  should have  received a  --
--  copy of the GNU General Public License  distributed  with  GNARL;  see  --
--  file  COPYING.  If not,  write to  the  Free  Software  Foundation, 59  --
--  Temple Place - Suite 330, Boston, MA 02111-1307, USA.                   --
--                                                                          --
--  Under contract  GS-35F-4506G, the U. S. Government obtained  unlimited  --
--  rights in the software and documentation contained herein.   Unlimited  --
--  rights are defined in DFAR 252,227-7013(a)(19).  By making this public  --
--  release,   the  Government  intends  to  confer  upon  all  recipients  --
--  unlimited  rights equal to those held by the Government.  These rights  --
--  include rights to use,  duplicate,  release  or  disclose the released  --
--  data an computer software  in whole or in part,  in any manner and for  --
--  any purpose whatsoever, and to have or permit others to do so.          --
--                                                                          --
--  DISCLAIMER   --   ALL MATERIALS OR INFORMATION HEREIN RELEASED,   MADE  --
--  AVAILABLE OR DISCLOSED ARE AS IS.   THE GOVERNMENT MAKES NO EXPRESS OR  --
--  IMPLIED WARRANTY AS TO ANY MATTER WHATSOEVER, INCLUDING THE CONDITIONS  --
--  OF THE SOFTWARE,  DOCUMENTATION  OR  OTHER INFORMATION RELEASED,  MADE  --
--  AVAILABLE OR DISCLOSED,  OR THE OWNERSHIP,  MERCHANTABILITY, OR FITNESS  --
--  FOR A PARTICULAR PURPOSE OF SAID MATERIAL.                              --
--                                                                          --
------------------------------------------------------------------------------

--  Basic test for package POSIX_Sockets_Internet,
--  in IEEE Std 1003.5c Section D.1.3

--  This test covers only features that depend only on
--  the package itself and features from other packages
--  that are required to be supported.

with POSIX,
     POSIX_Sockets,
     POSIX_Sockets_Internet,
     POSIX_Report,
     POSIX_IO,
     System,
     Unchecked_Conversion,
     Test_Parameters;

procedure pdd0100 is

   use POSIX,
       POSIX_Sockets,
       POSIX_Sockets_Internet,
       POSIX_Report;

--  ----- Procedure and Function Prototypes --------------------------  --

   procedure Action
      (Alias_Name : in     POSIX_String;
       Quit       : in out Boolean);

   generic
      type T is (<>);
         with function Get (Socket : POSIX_IO.File_Descriptor)
            return T;
         with procedure Set (Socket : in POSIX_IO.File_Descriptor;
            To : in T);
   procedure TCP_Tests (Name : in String; To : in T; Default : in T;
      Er0 : in String := "A000"; Er1, Er2, Er3 : in String);

   generic
      type T is (<>);
         with function Get (Socket : POSIX_IO.File_Descriptor)
            return T;
         with procedure Set (Socket : in POSIX_IO.File_Descriptor;
            To : in T);
   procedure IP_Tests (Name : in String; To : in T; Default : in T;
      Er0 : in String := "A000"; Er1, Er2 : in String);

--  ---- Procedure and Function Body ---------------------------------  --

   procedure Action
      (Alias_Name : in     POSIX_String;
       Quit       : in out Boolean)
   is
   begin
      if Alias_Name = "" then
         Quit := True;
      end if;
      Comment ("Alias name is : " & To_String (Alias_Name));
   end Action;

   procedure TCP_Tests (Name : in String; To : in T; Default : in T;
      Er0 : in String := "A000"; Er1, Er2, Er3 : in String)
   is
      Socket : POSIX_IO.File_Descriptor;
      Result : T;
   begin
      Test ("Get/Set_" & Name &" [D.1.3.4]");
      Socket := Create (Internet_Protocol, Stream_Socket, TCP);
      if Er0 /= "A000" then
         Assert (Get (Socket) = Default, Er0);
      end if;
      Set (Socket, To);
      Comment ("Socket's " & Name & " has been set");
      Result := Get (Socket);
      Assert (Result = To, Er1);
   exception
```

```
   when E1 : POSIX_Error =>
      Optional (Internet_Stream_Option, Unknown_Protocol_Option,
         E1, Er2);
   when E2 : others => Unexpected_Exception (E2, Er3);
   end TCP_Tests;

   procedure IP_Tests (Name : in String; To : in T; Default : in T;
      Er0 : in String := "A000"; Er1, Er2 : in String)
   is
      Socket : POSIX_IO.File_Descriptor;
      Result : T;
   begin
      Test ("Get/Set_" & Name &" [D.1.3.6]");
      Socket := Create (Internet_Protocol, Raw_Socket, Raw);
      if Er0 /= "A000" then
         Assert (Get (Socket) = Default, Er0);
      end if;
      Set (Socket, To);
      Comment ("Socket's " & Name & " has been set");
      Result := Get (Socket);
      Assert (Result = To, Er1);
   exception
   when E : others => Unexpected_Exception (E, Er2);
   end IP_Tests;

-------------------------------------------------------------------------
--  Begin Tests

begin
   Header ("pdd0100");
   Test ("package POSIX.Sockets.Internet");

   -----------------------------------------------------------------------
   --  Internet Socket Address Pointer arithmatic + operators return
   --  the proper types.

   Test ("Internet Socket Address [D.1.3.1]");
   declare
      Int_Ptr : Internet_Socket_Address_Pointer := null;
      Soc_Ptr : Socket_Address_Pointer := Null_Socket_Address;
   begin
      Soc_Ptr := +Int_Ptr;
      Int_Ptr := +Soc_Ptr;
      Assert (Int_Ptr = null, "A007");
   exception
   when E1 : POSIX_Error =>
      Optional (Sockets_DNI_Option, Internet_Protocol_Option,
         No_Error, E1, "A008");
   when E2 : others => Unexpected_Exception (E2, "A009");
   end;

   -----------------------------------------------------------------------
   --  Internet Port consants are correct and the Get/Set Internet_Port
   --  methods are consistent.

   Test ("Internet Port (Get/Set) [D.1.3.1]");
   declare
      Name : Internet_Socket_Address;
      Port : Internet_Port := 21; --  ftp port
   begin
      Assert (Unspecified_Internet_Port = 0, "A010");
      Set_Internet_Port (Name, 23); -- telnet port
      Comment ("Internet port set to 23");
      Port := Get_Internet_Port (Name);
      Assert (Port = 23, "A011");
   exception
   when E1 : POSIX_Error =>
      Optional (Sockets_DNI_Option, Internet_Protocol_Option,
         No_Error, E1, "A012");
   when E2 : others => Unexpected_Exception (E2, "A013");
   end;

   -----------------------------------------------------------------------
   --  String_To_Internet_Address and Internet_Address_To_String are
   --  consistent.  Is_Internet_Address returns the appropriate boolean
   --  value.
   --  This section is out of order because it is needed to test methods
   --  that occur in an earlier section.

   Test ("String/Internet_Address conversions [D.1.3.2]");
   declare
      type POSIX_String_Ptr is access POSIX_String;
      Name    : POSIX_String_Ptr;
      Address : Internet_Address;
   begin
      Assert (Is_Internet_Address ("555.1280.367.666") = False,
         "A014");
      Assert (Is_Internet_Address
              (Test_Parameters.Valid_Internet_Address) = True,
         "A015");
      Address := String_To_Internet_Address
        (Test_Parameters.Valid_Internet_Address);
      Comment ("Internet address set to " &
         To_String (Test_Parameters.Valid_Internet_Address));
      Name := new POSIX_String'(Internet_Address_To_String (Address));
      Comment ("Name = " & To_String (Name.all));
      Assert (Name.all = Test_Parameters.Valid_Internet_Address, "A016");
   exception
   when E1 : POSIX_Error =>
      Optional (Network_Management_Option,
         No_Error, E1, "A017");
   when E2 : others => Unexpected_Exception (E2, "A018");
   end;

   -----------------------------------------------------------------------
   --  Get/Set Internet_Address methods are consistent.

   Test ("Internet Port (Get/Set) [D.1.3.1]");
   declare
      Name    : Internet_Socket_Address;
      Address : Internet_Address;
   begin
      Address :=
        String_To_Internet_Address (Test_Parameters.Valid_Internet_Address);
      Set_Internet_Address (Name, Address);
      Comment ("Internet address set to "
         & To_String (Test_Parameters.Valid_Internet_Address));
      Address := Get_Internet_Address (Name);
      Assert (Internet_Address_To_String (Address) =
         Test_Parameters.Valid_Internet_Address, "A019");
   exception
   when E1 : POSIX_Error =>
      Optional (Sockets_DNI_Option, Internet_Protocol_Option,
         No_Error, E1, "A020");
   when E2 : others => Unexpected_Exception (E2, "A021");
   end;

   -----------------------------------------------------------------------
   --  Is_Internet_Socket_Address returns true if the object is an
   --  Internet_Socket_Address.  The false case cannot be tested without
   --  the use of other protocols (will be done in a sperate test program)

   Test ("Is_Internet_Socket_Address [D.1.3.1]");
   declare
      Name    : aliased Internet_Socket_Address;
      Address : Internet_Address;
   begin
      Address :=
        String_To_Internet_Address (Test_Parameters.Valid_Internet_Address);
      Set_Internet_Address (Name, Address);
      Assert (Is_Internet_Socket_Address (+(Name'Unchecked_Access)),
         "A022");
   exception
   when E1 : POSIX_Error =>
      Optional (Sockets_DNI_Option, Internet_Protocol_Option,
         No_Error, E1, "A023");
   when E2 : others => Unexpected_Exception (E2, "A024");
   end;

   -----------------------------------------------------------------------
   --  Get_Socket_Name return the correct value.

   Test ("Get_Socket_Name [D.1.3.1]");
   declare
      Name_Ptr : Internet_Socket_Address_Pointer;
      Address  : Internet_Address;
      Message  : Socket_Message;
      Name     : Internet_Socket_Address;
   begin
      Name_Ptr := new Internet_Socket_Address;
      Address :=
        String_To_Internet_Address (Test_Parameters.Valid_Internet_Address);
      Set_Internet_Address (Name_Ptr.all, Address);
      Set_Socket_Name (Message, +Name_Ptr);
      Comment ("Socket Name Set");
      Name := Get_Socket_Name (Message);
      Address := Get_Internet_Address (Name);
      Assert (Internet_Address_To_String (Address) =
         Test_Parameters.Valid_Internet_Address, "A025");
   exception
   when E1 : POSIX_Error =>
      Optional (Sockets_DNI_Option, Internet_Protocol_Option,
         No_Error, E1, "A026");
   when E2 : others => Unexpected_Exception (E2, "A027");
   end;

   -----------------------------------------------------------------------
   --  Get_Address return the correct value.

   Test ("Get_Address [D.1.3.1]");
   declare
      Name      : Internet_Address;
      Address   : Internet_Socket_Address;
      Info_Item : Socket_Address_Info;
      Info      : Socket_Address_Info_List;
      type Socket_Address_Info_Ptr is access all Socket_Address_Info;
      Info_Ptr  : Socket_Address_Info_Ptr;
      function To_sock_addr_info_ptr is new Unchecked_Conversion
         (System.Address, Socket_Address_Info_Ptr);
   begin
      begin
         Set_Flags (Info_Item, Canonical_Name);
         Set_Family (Info_Item, Internet_Protocol);
         Set_Socket_Type (Info_Item, Stream_Socket);
         Set_Protocol_Number (Info_Item, TCP);
         Get_Socket_Address_Info (Test_Parameters.Valid_Internet_Address,
            "telnet", Info_Item, Info);
      exception
      when E : POSIX_Error => Optional (Network_Management_Option,
         No_Error, E, "A028");
      end;

      Info_Ptr := To_sock_addr_info_ptr (Info'Address);
      Address := Get_Address (Info_Ptr.all);
      Name := Get_Internet_Address (Address);
      Assert (Internet_Address_To_String (Name) =
         Test_Parameters.Valid_Internet_Address, "A029");
   exception
   when E1 : POSIX_Error =>
      Optional (Sockets_DNI_Option, Internet_Protocol_Option,
```

```
      No_Error, E1, "A030");
when E2 : others => Unexpected_Exception (E2, "A031");
end;

------------------------------------------------------------------------
--  String_To_Internet_Address accepts all the proper dot notations
--  and raises the appropriate error messages when an inapropreate
--  string is used

Test ("String_To_Internet_Address [D.1.3.2]");
declare
   type POSIX_String_Ptr is access POSIX_String;
   Name    : POSIX_String_Ptr;
   Address : Internet_Address;
begin
   --  3 dot notation tested previously, see above
   Address := String_To_Internet_Address ("128.186.31017");
   Name := new POSIX_String'(Internet_Address_To_String (Address));
   Assert (Name.all = "128.186.121.41", "A032");
   Address := String_To_Internet_Address ("128.12220713");
   Name := new POSIX_String'(Internet_Address_To_String (Address));
   Assert (Name.all = "128.186.121.41", "A033");
   Address := String_To_Internet_Address ("2159704361");
   Name := new POSIX_String'(Internet_Address_To_String (Address));
   Assert (Name.all = "128.186.121.41", "A034");
exception
when E1 : POSIX_Error =>
   Optional (Network_Management_Option, No_Error, E1, "A035");
when E2 : others => Unexpected_Exception (E2, "A036");
end;

------------------------------------------------------------------------
--  Get_Network_Info_By_Name/By_Address and
--  Get_Name/Family/Network_Number are consitent.

Test ("Get_Network_Info [D.1.3.2]");
declare
   Info    : Network_Info;
   Storage : Database_Array_Pointer := new Database_Array (1 .. 50);
begin
   Info := Get_Network_Info_By_Name ("arpanet", Storage);
   Assert (Get_Name (Info) = "arpanet", "A037");
   Assert (Get_Family (Info) = Internet_Protocol, "A038");
   Assert (Get_Network_Number (Info) = 10, "A039");
   Info := Get_Network_Info_By_Address (10, Internet_Protocol, Storage);
   Assert (Get_Name (Info) = "arpanet", "A040");
   Assert (Get_Family (Info) = Internet_Protocol, "A041");
   Assert (Get_Network_Number (Info) = 10, "A042");
   Info := Get_Network_Info_By_Name ("@#$%^&*()", Storage);
   Assert (Get_Name (Info) = "", "A043");
   Info := Get_Network_Info_By_Address (999, Internet_Protocol, Storage);
   Assert (Get_Name (Info) = "", "A044");
   begin
      Storage := new Database_Array (1 .. 1);
      Info := Get_Network_Info_By_Name ("arpanet", Storage);
   exception when Constraint_Error => null;
   end;
   begin
      Info := Get_Network_Info_By_Address (10, Internet_Protocol, Storage);
   exception when Constraint_Error => null;
   end;
exception
when E1 : POSIX_Error =>
   Optional (Network_Management_Option, Unknown_Protocol_Option,
      E1, "A047");
when E2 : others => Unexpected_Exception (E2, "A048");
end;

------------------------------------------------------------------------
--  For_Every_Network_Alias calls Action for every alias.
--  Action places the alias name in a Comment.

Test ("For_Every_Network_Alias [D.1.3.2]");
declare
   Info    : Network_Info;
   Storage : Database_Array_Pointer := new Database_Array (1 .. 50);
   procedure Every_Net_Alias is
      new For_Every_Network_Alias (Action);
begin
   Info := Get_Network_Info_By_Name ("arpanet", Storage);
   Every_Net_Alias (Info);
exception
when E1 : POSIX_Error =>
   Optional (Network_Management_Option, Unknown_Protocol_Option,
      E1, "A049");
when E2 : others => Unexpected_Exception (E2, "A050");
end;


------------------------------------------------------------------------
--  Open/Close_Network_Database_Connection open and closes a
--  connection to the network database.

Test ("Open/Close_Network_Database_Connection [D.1.3.2]");
declare
   Info    : Network_Info;
   Storage : Database_Array_Pointer := new Database_Array (1 .. 50);
begin
   Open_Network_Database_Connection (False);
   Info := Get_Network_Info_By_Name ("arpanet", Storage);
   Assert (Get_Name (Info) = "arpanet", "A051");
   Close_Network_Database_Connection;
   Open_Network_Database_Connection (False);
   Info := Get_Network_Info_By_Name ("loopback", Storage);
   Info := Get_Network_Info_By_Name ("arpanet", Storage);
   Assert (Get_Name (Info) = "arpanet", "A052");
   Close_Network_Database_Connection;
   Open_Network_Database_Connection (True);
   Info := Get_Network_Info_By_Name ("loopback", Storage);
   Info := Get_Network_Info_By_Name ("arpanet", Storage);
   Assert (Get_Name (Info) = "arpanet", "A053");
   Close_Network_Database_Connection;
exception
when E1 : POSIX_Error =>
   Optional (Network_Management_Option, Unknown_Protocol_Option,
      E1, "A054");
when E2 : others => Unexpected_Exception (E2, "A055");
end;

------------------------------------------------------------------------
--  Get_Protocol_Info_By_Name/Number and
--  Get_Name/Protocol_Number are consitent.

Test ("Get_Protocol_Info [D.1.3.2]");
declare
   Info    : Protocol_Info;
   Storage : Database_Array_Pointer := new Database_Array (1 .. 50);
begin
   Info := Get_Protocol_Info_By_Name ("TCP", Storage);
   Assert (Get_Name (Info) = "tcp", "A056");
   Assert (Get_Protocol_Number (Info) = 6, "A057");
   Info := Get_Protocol_Info_By_Number (6, Storage);
   Assert (Get_Name (Info) = "tcp", "A058");
   Assert (Get_Protocol_Number (Info) = 6, "A059");
   Info := Get_Protocol_Info_By_Name ("@#$%^&*()", Storage);
   Assert (Get_Name (Info) = "", "A060");
   Info := Get_Protocol_Info_By_Number (999, Storage);
   Assert (Get_Name (Info) = "", "A061");
   begin
      Storage := new Database_Array (1 .. 2);
      Info := Get_Protocol_Info_By_Name ("tcp", Storage);
   exception when Constraint_Error => null;
   end;
   begin
      Info := Get_Protocol_Info_By_Number (6, Storage);
   exception when Constraint_Error => null;
   end;
exception
when E1 : POSIX_Error =>
   Optional (Network_Management_Option, Unknown_Protocol_Option,
      E1, "A064");
when E2 : others => Unexpected_Exception (E2, "A065");
end;

------------------------------------------------------------------------
--  For_Every_Protocol_Alias calls Action for every alias.
--  Action places the alias name in a Comment.

Test ("For_Every_Protocol_Alias [D.1.3.2]");
declare
   Info    : Protocol_Info;
   Storage : Database_Array_Pointer := new Database_Array (1 .. 50);
   procedure Every_Proto_Alias is
      new For_Every_Protocol_Alias (Action);
begin
   Info := Get_Protocol_Info_By_Name ("tcp", Storage);
   Every_Proto_Alias (Info);
exception
when E1 : POSIX_Error =>
   Optional (Network_Management_Option, Unknown_Protocol_Option,
      E1, "A066");
when E2 : others => Unexpected_Exception (E2, "A067");
end;


------------------------------------------------------------------------
--  Open/Close_Protocol_Database_Connection open and closes a
--  connection to the protocol database.

Test ("Open/Close_Protocol_Database_Connection [D.1.3.2]");
declare
   Info    : Protocol_Info;
   Storage : Database_Array_Pointer := new Database_Array (1 .. 50);
begin
   Open_Protocol_Database_Connection (False);
   Info := Get_Protocol_Info_By_Name ("tcp", Storage);
   Assert (Get_Name (Info) = "tcp", "A068");
   Close_Protocol_Database_Connection;
   Open_Protocol_Database_Connection (False);
   Info := Get_Protocol_Info_By_Name ("upd", Storage);
   Info := Get_Protocol_Info_By_Name ("tcp", Storage);
   Assert (Get_Name (Info) = "tcp", "A069");
   Close_Protocol_Database_Connection;
   Open_Protocol_Database_Connection (True);
   Info := Get_Protocol_Info_By_Name ("udp", Storage);
   Info := Get_Protocol_Info_By_Name ("tcp", Storage);
   Assert (Get_Name (Info) = "tcp", "A070");
   Close_Protocol_Database_Connection;
exception
when E1 : POSIX_Error =>
   Optional (Network_Management_Option, Unknown_Protocol_Option,
      E1, "A071");
when E2 : others => Unexpected_Exception (E2, "A072");
end;

------------------------------------------------------------------------
--  TCP Constants and types

Test ("TCP Constants and types [D.1.3.4]");
begin
```

```
   Assert (Keep_Alive_Time'First = 1, "A073");
   Assert (Keep_Alive_Time'Last = POSIX.Seconds'Last, "A074");
   Assert (Socket_Retransmit_Time'First = -1, "A075");
   Assert (Socket_Retransmit_Time'Last = POSIX.Seconds'Last, "A076");
   Assert (Wait_Forever = -1, "A077");
   Assert (Retransmit_Time_Default = 0, "A078");
exception when E : others => Unexpected_Exception (E, "A079");
end;

------------------------------------------------------------------------
--  Get/Set_Keep_Alive_Interval are consistent.

declare
   procedure Keep_Alive_Interval_Test is new TCP_Tests
      (Keep_Alive_Time, Get_Keep_Alive_Interval,
       Set_Keep_Alive_Interval);
begin
   --  Default must be 7200 or greater, but don't know exact value
   Keep_Alive_Interval_Test ("Keep_Alive_Interval", 5, 7200,
      "A000", "A080", "A081", "A082");
end;

------------------------------------------------------------------------
--  Get/Set_No_Delay are consistent.

declare
   procedure No_Delay_Test is new TCP_Tests
      (Socket_Option_Value, Get_No_Delay, Set_No_Delay);
begin
   No_Delay_Test ("No_Delay", Enabled, Disabled,
      "A083", "A084", "A085", "A086");
end;

------------------------------------------------------------------------
--  Get/Set_Retransmit_Time_Maximum are consistent.

declare
   procedure Retransmit_Time_Maximum_Test is new TCP_Tests
      (Socket_Retransmit_Time, Get_Retransmit_Time_Maximum,
       Set_Retransmit_Time_Maximum);
begin
   Retransmit_Time_Maximum_Test ("Retransmit_Time_Maxmimum", 5,
      Retransmit_Time_Default,
      "A087", "A088", "A089", "A090");
end;

------------------------------------------------------------------------
--  Get/Set_Standardized_Urgent_Data are consistent.

declare
   procedure Standardized_Urgent_Data_Test is new TCP_Tests
      (Socket_Option_Value, Get_Standardized_Urgent_Data,
       Set_Standardized_Urgent_Data);
begin
   Standardized_Urgent_Data_Test ("Standardized_Urgent_Data",
      Enabled, Disabled, "A091", "A092", "A093", "A094");
end;

------------------------------------------------------------------------
--  Get_Segment_Size_Maximum returns the current segement size, in
--  octets, of the TCP connection.

Test ("Get_Segment_Size_Maximum [D.1.3.4]");
declare
   Socket : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Assert (Get_Segment_Size_Maximum (Socket) /= 0, "A095");
exception
when E1 : POSIX_Error =>
   Optional (Internet_Stream_Option, Unknown_Protocol_Option,
      E1, "A096");
when E2 : others => Unexpected_Exception (E2, "A097");
end;


-----------------------------------------------------------------------
--                                                                   --
--  Section D.1.3.5 does not contain any information that is         --
--  protocol dependend.                                              --
--                                                                   --
-----------------------------------------------------------------------


------------------------------------------------------------------------
--  IP Constants and types

Test ("IP Constants and types [D.1.3.6]");
begin
   Assert (Time_To_Live'First = 0, "A098");
   Assert (Time_To_Live'Last = 255, "A099");
exception when E : others => Unexpected_Exception (E, "A100");
end;

------------------------------------------------------------------------
--  Get_First_Hop returns Unspecified_Internet_Address no
--  matter what Set_First_Hop sets it to if the Source Route iption
--  is not used.

Test ("Set/Get_First_Hop [D.1.3.6]");
declare
   Options : IP_Options_Buffer;
begin
   Set_First_Hop (Options,
                  String_To_Internet_Address
                    (Test_Parameters.Valid_Internet_Address));
   Comment ("First Hop Set");
   Assert (Get_First_Hop (Options) =
           Unspecified_Internet_Address, "A101");
exception
when E : others => Unexpected_Exception (E, "A102");
end;

------------------------------------------------------------------------
--  Get/Set_IP_Options are consistent.

Test ("Set/Get_IP_Options [D.1.3.6]");
declare
   Options : IP_Options_Buffer;
   Buffer  : POSIX.Octet_Array (1 .. 40);
begin
   Buffer (1 .. 4) := (116, 101, 115, 116);
   Set_IP_Options (Options, Buffer);
   Comment ("IP Options Set");
   Assert (Get_IP_Options (Options) = Buffer, "A103");
exception
when E : others => Unexpected_Exception (E, "A104");
end;

------------------------------------------------------------------------
--  Get/Set_IP_Header_Options are consistent.
--  IP_Header_Options_In_Use returns true when header options are in
--    use and flase otherwise.
--  Reset_IP_Header_Options removes any header options that are in
--    effect.

Test ("IP_Header_Options [D.1.3.6]");
declare
   Options, Result : IP_Options_Buffer;
   Buffer  : POSIX.Octet_Array (1 .. 40);
   Socket  : POSIX_IO.File_Descriptor;
begin
   Buffer (1 .. 4) := (116, 101, 115, 116);
   Set_IP_Options (Options, Buffer);
   Set_First_Hop (Options,
      String_To_Internet_Address
              (Test_Parameters.Valid_Internet_Address));
   Socket := Create (Internet_Protocol, Raw_Socket);
   Assert (IP_Header_Options_In_Use (Socket) = False, "A105");
   Set_IP_Header_Options (Socket, Options);
   Result := Get_IP_Header_Options (Socket);
   Assert (Options = Result, "A106");
   Assert (IP_Header_Options_In_Use (Socket) = True, "A107");
   Reset_IP_Header_Options (Socket);
   Assert (IP_Header_Options_In_Use (Socket) = False, "A108");
exception
when E : others => Unexpected_Exception (E, "A109");
end;

------------------------------------------------------------------------
--  Get/Set_Type_Of_Service are consistent.

Test ("Get/Set_Type_Of_Service [D.1.3.6]");
declare
   Socket : POSIX_IO.File_Descriptor;
   Result : IP_Type_Of_Service;
begin
   Socket := Create (Internet_Protocol, Raw_Socket, Raw);
   --  Default value is Unspecified
   Assert (Get_Type_Of_Service (Socket) =
      Unspecified, "A110");
   Set_Type_Of_Service (Socket, High_Throughput);
   Comment ("Socket's Type_Of_Service has been set");
   Result := Get_Type_Of_Service (Socket);
   Assert (Result = High_Throughput, "A111");
exception
when E : others => Unexpected_Exception (E, "A112");
end;

------------------------------------------------------------------------
--  Get/Set_Initial_Time_To_Live are consistent.

declare
   procedure Initial_Time_To_Live_Test is new IP_Tests
      (Time_To_Live, Get_Initial_Time_To_Live,
       Set_Initial_Time_To_Live);
begin
   --  Default is implementation specific
   Initial_Time_To_Live_Test ("Intial_Time_To_Live", 5, 0,
      "A000", "A113", "A114");
end;

------------------------------------------------------------------------
--  Get/Set_Receive_Destination_Address are consistent.

declare
   procedure Receive_Destination_Address_Test is new IP_Tests
      (Socket_Option_Value, Get_Receive_Destination_Address,
       Set_Receive_Destination_Address);
begin
   --  Default value is Disabled
   Receive_Destination_Address_Test ("Receive_Destination_Address",
      Enabled, Disabled, "A115", "A116", "A117");
end;

------------------------------------------------------------------------
--                                                                    --
--  Set_Ancillary_Data and Get_Destination_Address cannot be properly --
--  tested without an application.  Therefore their testing will be   --
--  done in one of the p1804 series of tests.                         --
```

```
--                                                                        --
----------------------------------------------------------------------------

   -------------------------------------------------------------------------
   --  Get/Set_Type_Of_Service are consistent.

   Test ("Get/Set_Header_Included [D.1.3.6]");
   declare
      Socket : POSIX_IO.File_Descriptor;
      Result : Socket_Option_Value;
   begin
      --  Currently cannot create a socket of type Raw
      Socket := Create (Internet_Protocol, Raw_Socket, Raw);
      --  Default value is Disabled
      Assert (Get_Header_Included (Socket) = Disabled, "A118");
      Set_Header_Included (Socket, Enabled);
      Comment ("Socket's Header_Included has been enabled");
      Result := Get_Header_Included (Socket);
      Assert (Result = Enabled, "A119");
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Option_Not_Supported then
         Unexpected_Exception (E1, "A120");
      end if;
   when E2 : others => Unexpected_Exception (E2, "A121");
   end;

   Done;
exception when E : others => Fatal_Exception (E, "A122");
end pdd0100;
```

p180401.adb

```
------------------------------------------------------------------------------
--                                                                          --
--                      POSIX.5c VALIDATION TEST SUITE                      --
--                                                                          --
--                             P 1 8 0 4 0 1                                --
--                                                                          --
--                                B o d y                                   --
--                                                                          --
--                                                                          --
--  Copyright (c) 1995-1999 Florida  State  University  (FSU).  All Rights  --
--  Reserved.                                                               --
--                                                                          --
--  This is free software;  you can redistribute it and/or modify it under  --
--  terms of the  GNU  General  Public  License  as published by the  Free  --
--  Software Foundation;  either version 2, or (at your option) any  later  --
--  version.  This  software  is distributed  in the hope that it  will be  --
--  useful, but WITHOUT ANY WARRANTY; without even the implied warranty of  --
--  MERCHANTABILITY   or  FITNESS FOR A PARTICULAR PURPOSE.   See the  GNU  --
--  General Public License for more details.  You  should have  received a  --
--  copy of the GNU General Public License  distributed  with  GNARL;  see  --
--  file  COPYING.  If not,  write to  the  Free  Software  Foundation, 59  --
--  Temple Place - Suite 330, Boston, MA 02111-1307, USA.                   --
--                                                                          --
--  Under contract  GS-35F-4506G, the U. S. Government obtained  unlimited  --
--  rights in the software and documentation contained herein.   Unlimited  --
--  rights are defined in DFAR 252,227-7013(a)(19).  By making this public  --
--  release,   the  Government  intends  to  confer  upon  all  recipients  --
--  unlimited  rights equal to those held by the Government.  These rights  --
--  include rights to use,  duplicate,  release  or  disclose the released  --
--  data an computer software  in whole or in part,  in any manner and for  --
--  any purpose whatsoever, and to have or permit others to do so.          --
--                                                                          --
--  DISCLAIMER   --   ALL MATERIALS OR INFORMATION HEREIN RELEASED,   MADE  --
--  AVAILABLE OR DISCLOSED ARE AS IS.   THE GOVERNMENT MAKES NO EXPRESS OR  --
--  IMPLIED WARRANTY AS TO ANY MATTER WHATSOEVER, INCLUDING THE CONDITIONS  --
--  OF THE SOFTWARE,  DOCUMENTATION  OR  OTHER INFORMATION RELEASED,  MADE  --
--  AVAILABLE OR DISCLOSED,  OR THE OWNERSHIP,  MERCHANTABILITY, OR FITNESS --
--  FOR A PARTICULAR PURPOSE OF SAID MATERIAL.                              --
--                                                                          --
------------------------------------------------------------------------------

--  Integrated test for package POSIX_Sockets
--  in IEEE Std 1003.5c Section 18.4 with package
--  POSIX_Sockets_Interent.

--  This test covers only features that depend only on
--  the packages (POSIX_Sockets/_Interent) and features from
--  other packages that are required to be supported.

with POSIX,
     POSIX_Sockets,
     POSIX_Sockets_Internet,
     POSIX_Report,
     POSIX_IO,
     Unchecked_Conversion,
     System,
     Test_Parameters;

procedure p180401 is

   use POSIX,
       POSIX_Sockets,
       POSIX_Sockets_Internet,
       POSIX_IO,
       POSIX_Report,
       Test_Parameters;

   procedure Get_Socket_Address_Info_4Param_Tests (
      Name     : in POSIX_String    := Valid_Internet_Name;
      Service  : in POSIX_String    := "telnet";
      Flags    : in Address_Flags   := Use_For_Binding;
      Family   : in Protocol_Family := Internet_Protocol;
      To       : in Socket_Type     := Stream_Socket;
      Protocol : in Protocol_Number := TCP;
      Expected : in Error_Code      := Operation_Not_Implemented;
      Error1   : in String;
      Error2   : in String;
      Error3   : in String          := "");

generic
   type T is (<>);
      with function Get (Socket : in POSIX_IO.File_Descriptor)
         return T;
      with procedure Set
         (Socket : in POSIX_IO.File_Descriptor;
          To     : in T);
procedure Testit
   (Name     : in String;
    Init     : in T;
    Value    : in T;
    Er1, Er2, Er3, Er4, Er5, Er6, Er7, Er8, Er9,
      Er10 : in String);

procedure Create_Tests (
   Family   : in Protocol_Family := Internet_Protocol;
   Soc_Type : in Socket_Type     := Stream_Socket;
   Protocol : in Protocol_Number := TCP;
   Expected : in Error_Code      := No_Error;
   Error1   : in String;
   Error2   : in String          := "";
   Error3   : in String;
   Error4   : in String);

procedure Bind_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Port     : in Internet_Port
            := Unspecified_Internet_Port;
   Address  : in Internet_Address
            := Unspecified_Internet_Address;
   Expected : in Error_Code := No_Error;
   Error1, Error2, Error3 : in String);

procedure Connect_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Port     : in Internet_Port := 23;
   Address  : in Internet_Address
      := String_To_Internet_Address ("127.0.0.1");
   Expected : in Error_Code := No_Error;
   Error1, Error2, Error3 : in String);

procedure Specify_Peer_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Port     : in Internet_Port := 23;
   Address  : in Internet_Address
      := String_To_Internet_Address ("127.0.0.1");
   Expected : in Error_Code := No_Error;
   Error1, Error2, Error3 : in String);

procedure Unspecify_Peer_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Expected : in Error_Code := No_Error;
   Error1, Error2, Error3 : in String);

procedure Listen_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Backlog  : in Connection_Queue_Length
      := Connection_Queue_Length'Last;
   Expected : in Error_Code := No_Error;
   Error1, Error2, Error3 : in String;
   Close_Socket : in boolean := false);

procedure Shutdown_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Mode     : in Shutdown_Mode;
   Expected : in Error_Code := No_Error;
   Error1, Error2, Error3 : in String;
   Close_Socket : in boolean := false;
   Should_Connect : in boolean := true);

procedure Get_Socket_Address_Info_4Param_Tests (
   Name     : in POSIX_String    := Valid_Internet_Name;
   Service  : in POSIX_String    := "telnet";
   Flags    : in Address_Flags   := Use_For_Binding;
   Family   : in Protocol_Family := Internet_Protocol;
   To       : in Socket_Type     := Stream_Socket;
   Protocol : in Protocol_Number := TCP;
   Expected : in Error_Code      := Operation_Not_Implemented;
   Error1   : in String;
   Error2   : in String;
   Error3   : in String          := "")
is
   Info    : Socket_Address_Info_List;
   Request : Socket_Address_Info;
begin
   Set_Flags (Request, Flags);
   Set_Family (Request, Family);
   Set_Socket_Type (Request, To);
   Set_Protocol_Number (Request, Protocol);
   Get_Socket_Address_Info (Name, Service, Request, Info);
   if Error3 /= "" then
      Expect_Exception (Error3);
   end if;
exception
```

```
when E1 : POSIX_Error =>
   if Get_Error_Code /= Expected then
      Optional (Network_Management_Option,
         Unknown_Protocol_Option, E1, Error1);
   end if;
when E2 : others => Unexpected_Exception (E2, Error2);
end Get_Socket_Address_Info_4Param_Tests;

----------------------------------------------------------------------
--  Procedure used to test Get and Set methods used with protocol
--  independent Socket information

procedure Testit
   (Name     : in String;
    Init     : in T;
    Value    : in T;
    Er1, Er2, Er3, Er4, Er5, Er6, Er7, Er8, Er9,
       Er10  : in String)

is
begin
   -------------------------------------------------------------------
   --  The Set and Get methods are consistent

   Test ("Set/Get_" & Name & " [18.4.9]");
   declare
      Socket : POSIX_IO.File_Descriptor := 0;
      To     : T                        := Value;
   begin
      Socket := Create (Internet_Protocol, Stream_Socket, TCP);
      Set (Socket, Init);
      Comment ("Socket Set_" & Name);
      To := Get (Socket);
      Assert (To = Init, Er1);
   exception when E : others => Unexpected_Exception (E, Er2);
   end;

   -------------------------------------------------------------------
   --  The Get method returns a default value

   Test ("Get_" & Name & " (Default) [18.4.9.2]");
   declare
      Socket : POSIX_IO.File_Descriptor := 0;
      To     : T                        := Init;
   begin
      Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
      To := Get (Socket);
      Assert (To = Value, Er3);
   exception when E : others => Unexpected_Exception (E, Er4);
   end;

   -------------------------------------------------------------------
   --  The Set causes the Not_A_Socket POSIX error to
   --  be raised when the socket has not been created

   Test (Name & ": Not_A_Socket [18.4.9.3]");
   declare
      Socket : POSIX_IO.File_Descriptor := 0;
      To     : T                        := Init;
   begin
      Set  (Socket, To);
      Expect_Exception (Er5);
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Not_A_Socket then
         Unexpected_Exception (E1, Er6);
      end if;
   when E2 : others => Unexpected_Exception (E2, Er7);
   end;

   -------------------------------------------------------------------
   --  The Get  causes the Not_A_Socket POSIX error to
   --  be raised when the socket has not been created

   Test (Name & ": Not_A_Socket [18.4.9.3]");
   declare
      Socket : POSIX_IO.File_Descriptor := 0;
      To     : T                        := Init;
   begin
      To := Get (Socket);
      Expect_Exception (Er8);
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Not_A_Socket then
         Unexpected_Exception (E1, Er9);
      end if;
   when E2 : others => Unexpected_Exception (E2, Er10);
   end;
end Testit;

procedure Create_Tests (
   Family   : in Protocol_Family := Internet_Protocol;
   Soc_Type : in Socket_Type     := Stream_Socket;
   Protocol : in Protocol_Number := TCP;
   Expected : in Error_Code      := No_Error;
   Error1   : in String;
   Error2   : in String          := "";
   Error3   : in String;
   Error4   : in String)
is
   Sock : POSIX_IO.File_Descriptor;
begin
   Sock := Create (Family, Soc_Type, Protocol);
   if Error2 /= "" then
      Expect_Exception (Error2);
   end if;
   Close (Sock);
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Expected then
      Unexpected_Exception (E1, Error3);
   end if;
when E2 : others =>
   Unexpected_Exception (E2, Error4);
end Create_Tests;

procedure Bind_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Port     : in Internet_Port
            := Unspecified_Internet_Port;
   Address  : in Internet_Address
            := Unspecified_Internet_Address;
   Expected : in Error_Code := No_Error;
   Error1, Error2, Error3 : in String)
is
   Int_Add : aliased Internet_Socket_Address;
   Name    : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, Port);
   Set_Internet_Address (Int_Add, Address);
   Bind (Socket, +Name);
   if Error1 /= "" then
      Expect_Exception (Error1);
   end if;
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Expected then
      Unexpected_Exception (E1, Error2);
   end if;
when E2 : others => Unexpected_Exception (E2, Error3);
end Bind_Tests;

procedure Connect_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Port     : in Internet_Port := 23;
   Address  : in Internet_Address
      := String_To_Internet_Address ("127.0.0.1");
   Expected : in Error_Code := No_Error;
   Error1, Error2, Error3 : in String)
is
   Int_Add : aliased Internet_Socket_Address;
   Name    : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, Port);
   Set_Internet_Address (Int_Add, Address);
   Connect (Socket, +Name);
   if Error1 /= "" then
      Expect_Exception (Error1);
   end if;
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Expected then
      Unexpected_Exception (E1, Error2);
   end if;
when E2 : others => Unexpected_Exception (E2, Error3);
end Connect_Tests;

procedure Specify_Peer_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Port     : in Internet_Port := 23;
   Address  : in Internet_Address
      := String_To_Internet_Address ("127.0.0.1");
   Expected : in Error_Code := No_Error;
   Error1, Error2, Error3 : in String)
is
   Int_Add : aliased Internet_Socket_Address;
   Name    : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, Port);
   Set_Internet_Address (Int_Add, Address);
   Specify_Peer (Socket, +Name);
   if Error1 /= "" then
      Expect_Exception (Error1);
   end if;
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Expected then
      Unexpected_Exception (E1, Error2);
   end if;
when E2 : others => Unexpected_Exception (E2, Error3);
end Specify_Peer_Tests;

procedure Unspecify_Peer_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Expected : in Error_Code := No_Error;
   Error1, Error2, Error3 : in String)
is
begin
   Unspecify_Peer (Socket);
   if Error1 /= "" then
      Expect_Exception (Error1);
   end if;
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Expected then
      Unexpected_Exception (E1, Error2);
   end if;
```

```
   when E2 : others => Unexpected_Exception (E2, Error3);
   end Unspecify_Peer_Tests;

   procedure Listen_Tests (
      Socket   : in POSIX_IO.File_Descriptor;
      Backlog  : in Connection_Queue_Length
         := Connection_Queue_Length'Last;
      Expected : in Error_Code := No_Error;
      Error1, Error2, Error3 : in String;
      Close_Socket : in boolean := false)
   is
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
         Int_Add'Unchecked_Access;
   begin
      Set_Internet_Port (Int_Add, Unspecified_Internet_Port);
      Set_Internet_Address (Int_Add,
         String_To_Internet_Address ("127.0.0.1"));
      Bind (Socket, +Name);
      if Close_Socket = true then
         Close (Socket);
      end if;
      Listen (Socket, Backlog);
      if Error1 /= "" then
         Expect_Exception (Error1);
      end if;
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Expected then
         Unexpected_Exception (E1, Error2);
      end if;
   when E2 : others => Unexpected_Exception (E2, Error3);
   end Listen_Tests;

   procedure Shutdown_Tests (
      Socket   : in POSIX_IO.File_Descriptor;
      Mode     : in Shutdown_Mode;
      Expected : in Error_Code := No_Error;
      Error1, Error2, Error3 : in String;
      Close_Socket : in boolean := false;
      Should_Connect : in boolean := true)
   is
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
         Int_Add'Unchecked_Access;
   begin
      Set_Internet_Port (Int_Add, 23);
      Set_Internet_Address (Int_Add,
         String_To_Internet_Address ("127.0.0.1"));
      if Should_Connect = true then
         Connect (Socket, +Name);
      end if;
      if Close_Socket = true then
         Close (Socket);
      end if;
      Shutdown (Socket, Mode);
      if Error1 /= "" then
         Expect_Exception (Error1);
      end if;
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Expected then
         Unexpected_Exception (E1, Error2);
      end if;
   when E2 : others => Unexpected_Exception (E2, Error3);
   end Shutdown_Tests;


-------------------------------------------------------------------------
--  Begin Tests

begin
   Header ("p180401");
   Test ("package POSIX.Sockets");

   ----------------------------------------------------------------------
   --  If the Network Management option is supported then Info is filled
   --  up with the appropriate information

   Test ("Get_Socket_Address_Info (4-param version) [18.4.7]");
   begin
      Get_Socket_Address_Info_4Param_Tests
         (Error1 => "A090", Error2 => "A091");
   end;

   ----------------------------------------------------------------------
   --  If the Network Management option is supported then Info is filled
   --  up with the appropriate information

   Test ("Get_Socket_Address_Info (4-param with null Name) [18.4.7]");
   begin
      Get_Socket_Address_Info_4Param_Tests
         ("", Error1 => "A092", Error2 => "A093");
   end;

   ----------------------------------------------------------------------
   --  If the Network Management option is supported then Info is filled
   --  up with the appropriate information

   Test ("Get_Socket_Address_Info (4-param with null Service) [18.4.7]");
   begin
      Get_Socket_Address_Info_4Param_Tests
         (Service => "", Error1 => "A094", Error2 => "A095");
   end;

   ----------------------------------------------------------------------
   --  If the Network Management option is supported then the
   --  Service_Not_Supported POSIX_Error code will be raised since the
   --  socket type doesn't have the requested service

   Test (
    "Get_Socket_Address_Info (4-param): Service_Not_Supported [18.4.7]");
   begin
      Get_Socket_Address_Info_4Param_Tests
         (Valid_Internet_Name, "retelnet",
          Expected => Service_Not_Supported,
          Error1 => "A099", Error2 => "A100", Error3 => "A101");
   end;

   ----------------------------------------------------------------------
   --  If the Network Management option is supported then the
   --  Name_Not_Known POSIX_Error code will be raised

   Test ("Get_Socket_Address_Info (4-param): Name_Not_Known [18.4.7]");
   begin
      Get_Socket_Address_Info_4Param_Tests
         (Unknown_Name, Expected => Name_Not_Known,
          Error1 => "A102", Error2 => "A103", Error3 => "A104");
   end;

   ----------------------------------------------------------------------
   --  If the Network Management option is supported then the
   --  Invalid_Flags POSIX_Error code should be raised

   --  Test removed bacuase an invalid set of flags could not
   --  be determined.

   ----------------------------------------------------------------------
   --  If the Network Management option is supported then the
   --  Unknown_Protocol_Family POSIX_Error code will be raised

   --  Test removed due to unknown cause of STORAGE_ERROR.  It was
   --  working fine with Family Protocol of 19 but started giveing
   --  STORAGE_ERRORS for no explainable reason.

   ----------------------------------------------------------------------
   --  A Socket properly filled in is created

   Test ("Create [18.4.5]");
   begin
      Create_Tests (Error1 => "A021", Error3 => "A022",
         Error4 => "A023");
   end;

   ----------------------------------------------------------------------
   --  A Socket that uses the default parameter is created

   Test ("Create (default) [18.4.5]");
   declare
      Sock : POSIX_IO.File_Descriptor := 0;
   begin
      Sock := Create (Internet_Protocol, Stream_Socket);
      Assert (Sock /= 0, "A024");
   exception when E : others => Unexpected_Exception (E, "A025");
   end;

   ----------------------------------------------------------------------
   --  A bad protocol family creates the Protocol_Not_Supported
   --  POSIX_Error code

   Test ("Create: Protocol_Not_Supported [18.4.5.3]");
   begin
      Create_Tests (Family => Unknown_Family,
      Expected => Protocol_Not_Supported, Error1 => "A026",
      Error2 => "A027", Error3 => "A028", Error4 => "A029");
   end;

   ----------------------------------------------------------------------
   --  A bad socket type causes a Socket_Not_Supported POSOIX_Error code

   --  Test removed because Socket_Not_Supported is superceded by
   --  Protocol_Not_Supported.  If a socket is incorrect for a correct
   --  protocol the protocol is interpreted as being wrong.

   ----------------------------------------------------------------------
   --  If the Network Management option is supported then the Get and Set
   --  methods are consistent

   Test ("Set/Get_Protocol_Number [18.4.7]");
   declare
      Info_Item : Socket_Address_Info;
      Protocol  : Protocol_Number;
   begin
      Protocol := TCP;
      Set_Protocol_Number (Info_Item, Protocol);
      Protocol := UDP;

      --  The Assert statement is used to garauntee that the optimizer
      --  doesn't remove the previous assignment due to it being dead code.
      Assert (Protocol = 17, "A069");
      Protocol := Get_Protocol_Number (Info_Item);
      Assert (Protocol = 6, "A070");
   exception
   when E1 : POSIX_Error =>
      Optional (Network_Management_Option,
         Unknown_Protocol_Option, E1, "A071");
   when E2 : others => Unexpected_Exception (E2, "A072");
   end;

   ----------------------------------------------------------------------
```

```
--  If the Network Management option is supported then the get and set
--  methods are consistent

Test ("Set/Get_Family Test [18.4.7]");
declare
   Info_Item : Socket_Address_Info;
   Family    : Protocol_Family;
begin
   Family := Internet_Protocol;
   Set_Family (Info_Item, Family);
   Family := Inappro_Family;

   --  The Assert statement is used to garauntee that the optimizer
   --  doesn't remove the previous assignment due to it being dead code.
   Assert (Family = Inappro_Family, "A061");
   Family := Get_Family (Info_Item);
   Assert (Family = Internet_Protocol, "A062");
exception
when E1 : POSIX_Error =>
   Optional (Network_Management_Option,
      Unknown_Protocol_Option, E1, "A063");
when E2 : others => Unexpected_Exception (E2, "A064");
end;

-----------------------------------------------------------------------
--  If the socket has a pending error it will be returned and the
--  error status shall be set to 0

Test ("Get_Socket_Error_Status  [18.4.8]");
declare
   Sock  : POSIX_IO.File_Descriptor := 0;
   Error : POSIX.Error_Code := 0;
begin
   Sock := Create (Internet_Protocol, Stream_Socket, TCP);
   Error := Get_Socket_Error_Status (Sock);
   if Error /= 0 then
      Comment ("Socket returned and error; Expected");
   end if;
exception when E : others => Unexpected_Exception (E, "A114");
end;

-----------------------------------------------------------------------
--  A socket type of Stream_Socket will be returned

Test ("Get_Socket_Type  [18.4.8]");
declare
   Sock           : POSIX_IO.File_Descriptor := 0;
   Type_Of_Socket  : Socket_Type := Datagram_Socket;
begin
   Sock := Create (Internet_Protocol, Stream_Socket, TCP);
   Type_Of_Socket := Get_Socket_Type (Sock);
   Assert (Type_Of_Socket = Stream_Socket, "A118");
exception when E : others => Unexpected_Exception (E, "A119");
end;

-----------------------------------------------------------------------
--  Since the socket was never created an error of Not_A_Socket will
--  be raised

Test ("Get_Socket_Type: Not_A_Socket  [18.4.8]");
declare
   Sock           : POSIX_IO.File_Descriptor := 0;
   Type_Of_Socket  : Socket_Type := Datagram_Socket;
begin
   Type_Of_Socket := Get_Socket_Type (Sock);
   Assert (Type_Of_Socket = Stream_Socket, "A120");
   Expect_Exception ("A121");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Not_A_Socket then
      Unexpected_Exception (E1, "A122");
   end if;
when E2 : others => Unexpected_Exception (E2, "A123");
end;

-----------------------------------------------------------------------
--  The Set and Get_Socket_Receive_Buffer_Size methods are consistent
--  Doesn't work with a conenction mode socket
--  Has to many special cases to use generic

Test ("Set/Get_Socket_Receive_Buffer_Size [18.4.9]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count := 16;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Set_Socket_Receive_Buffer_Size (Socket, 32);
   Comment ("Socket receive buffer size set to 32");
   To := Get_Socket_Receive_Buffer_Size (Socket);
   Assert (To = 32, "A201");
exception when E : others => Unexpected_Exception (E, "A202");
end;

-----------------------------------------------------------------------
--  The Get_Socket_Receive_Buffer_Size method has a protocol defined
--  default value, wich is verly unlikly to be 0

Test ("Get_Socket_Receive_Buffer_Size (Default) [18.4.9.2]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count := 0;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   To := Get_Socket_Receive_Buffer_Size (Socket);
   Assert (To /= 0, "A203");
exception when E : others => Unexpected_Exception (E, "A204");
end;

-----------------------------------------------------------------------
--  The Set_Socket_Receive_Buffer_Size causes the Not_A_Socket
--  POSIX error to be raised when the socket has not been created

Test ("Set_Socket_Receive_Buffer_Size: Not_A_Socket [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count           := 256;
begin
   Set_Socket_Receive_Buffer_Size (Socket, To);
   Expect_Exception ("A205");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Not_A_Socket then
      Unexpected_Exception (E1, "A206");
   end if;
when E2 : others => Unexpected_Exception (E2, "A207");
end;

-----------------------------------------------------------------------
--  The Get_Socket_Receive_Buffer_Size causes the Not_A_Socket
--  POSIX error to be raised when the socket has not been created

Test ("Get_Socket_Receive_Buffer_Size: Not_A_Socket [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count           := 256;
begin
   To := Get_Socket_Receive_Buffer_Size (Socket);
   Expect_Exception ("A208");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Not_A_Socket then
      Unexpected_Exception (E1, "A209");
   end if;
when E2 : others => Unexpected_Exception (E2, "A210");
end;

-----------------------------------------------------------------------
--  The Set_Socket_Receive_Buffer_Size causes a No_Buffer_Space
--  Error when the requested buffer is too big

Test ("Set_Socket_Receive_Buffer_Size: No_Buffer_Space [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
   To     : POSIX.IO_Count           := 10000000;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Set_Socket_Receive_Buffer_Size (Socket, To);
   Expect_Exception ("A213");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= No_Buffer_Space then
      Unexpected_Exception (E1, "A214");
   end if;
when E2 : others => Unexpected_Exception (E2, "A215");
end;

-----------------------------------------------------------------------
--  The Set and Get_Socket_Receive_Low_Water_Mark methods are
--  consistent if the implementation allows the value to be set.
--  Otherwise the Permission_Denied Error will be raised
--  If a system does not even support the Receive_Low_Water_Mark
--  then the Unknown_Protocol_Option error will be raised

Test ("Set/Get_Socket_Receive_Low_Water_Mark [18.4.9]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count := 2;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Set_Socket_Receive_Low_Water_Mark (Socket, 3);
   Comment ("Socket receive low water mark set to 3");
   To := Get_Socket_Receive_Low_Water_Mark (Socket);
   Assert (To = 3, "A216");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code = Unknown_Protocol_Option then
      Comment ("Unknown_Protocol_Option");
   else
      Unexpected_Exception (E1, "A217");
   end if;
when E2 : others => Unexpected_Exception (E2, "A218");
end;

-----------------------------------------------------------------------
--  The Get_Socket_Receive_Low_Water_Mark method has a default value
--  of 1

Test ("Get_Socket_Receive_Low_Water_Mark (Default) [18.4.9.2]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count := 0;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   To := Get_Socket_Receive_Low_Water_Mark (Socket);
   Assert (To = 1, "A219");
exception
when POSIX_Error =>
   if Get_Error_Code = Unknown_Protocol_Option then
```

```
      Comment ("Unknown_Protocol_Option");
   end if;
when E : others => Unexpected_Exception (E, "A220");
end;

------------------------------------------------------------------------
--  The Set_Socket_Receive_Low_Water_Mark method causes the
--  Not_A_Socket POSIX error to be raised when the socket
--  has not been created (See above for other cases)

Test ("Set_Socket_Receive_Low_Water_Mark: Not_A_Socket [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count           := 2;
begin
   Set_Socket_Receive_Low_Water_Mark (Socket, To);
   Expect_Exception ("A221");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code /= Not_A_Socket then
      Unexpected_Exception (E1, "A222");
   end if;
when E2 : others => Unexpected_Exception (E2, "A223");
end;

------------------------------------------------------------------------
--  The Get_Socket_Receive_Low_Water_Mark method causes the
--  Not_A_Socket POSIX error to be raised when the socket has not
--  been created (see above for other cases)

Test ("Get_Socket_Receive_Low_Water_Mark: Not_A_Socket [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count           := 2;
begin
   To := Get_Socket_Receive_Buffer_Size (Socket);
   Expect_Exception ("A224");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code /= Not_A_Socket then
      Unexpected_Exception (E1, "A225");
   end if;
when E2 : others => Unexpected_Exception (E2, "A226");
end;

------------------------------------------------------------------------
--  The Set and Get_Socket_Receive_Timeout methods are consistent

Test ("Set/Get_Socket_Receive_Timeout [18.4.9]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : Duration := 16.0;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Set_Socket_Receive_Timeout (Socket, 32.0);
   Comment ("Socket receive timout set to 32");
   To := Get_Socket_Receive_Timeout (Socket);
   Assert (To = 32.0, "A229");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code = Unknown_Protocol_Option then
      Comment ("Unknown_Protocol_Option");
   else
      Unexpected_Exception (E1, "A230");
   end if;
when E2 : others => Unexpected_Exception (E2, "A231");
end;

------------------------------------------------------------------------
--  The Get_Socket_Receive_Timeout method has a defined
--  default value of 0

Test ("Get_Socket_Receive_Timeout (Default) [18.4.9.2]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : Duration                 := 2.0;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   To := Get_Socket_Receive_Timeout (Socket);
   Assert (To = 0.0, "A232");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code = Unknown_Protocol_Option then
      Comment ("Unknown_Protocol_Option");
   else
      Unexpected_Exception (E1, "A233");
   end if;
when E2 : others => Unexpected_Exception (E2, "A234");
end;

------------------------------------------------------------------------
--  The Set_Socket_Receive_Timeout causes the Not_A_Socket
--  POSIX error to be raised when the socket has not been created

Test ("Set_Socket_Receive_Timout: Not_A_Socket [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 1;
   To     : Duration                 := 3.0;
begin
   Set_Socket_Receive_Timeout (Socket, To);
   Expect_Exception ("A235");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code /= Not_A_Socket then
      Unexpected_Exception (E1, "A236");
   end if;
when E2 : others => Unexpected_Exception (E2, "A237");
end;

------------------------------------------------------------------------
--  The Get_Socket_Receive_Timeout causes the Not_A_Socket
--  POSIX error to be raised when the socket has not been created

Test ("Get_Socket_Receive_Timeout: Not_A_Socket [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 3;
   To     : Duration                 := 2.0;
begin
   To := Get_Socket_Receive_Timeout (Socket);
   Expect_Exception ("A238");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code /= Not_A_Socket then
      Unexpected_Exception (E1, "A239");
   end if;
when E2 : others => Unexpected_Exception (E2, "A240");
end;

------------------------------------------------------------------------
--  The Set_Socket_Receive_Timout causes a Domain_Error
--  Error when the requested buffer is too big

Test ("Set_Socket_Receive_Timeout: Domain_Error [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
   To     : Duration := 10000000.0;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Set_Socket_Receive_Timeout (Socket, To);
   Expect_Exception ("A243");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code = Unknown_Protocol_Option then
      Comment ("Unknown_Protocol_Option");
   elsif Get_Error_Code /= No_Buffer_Space then
      Unexpected_Exception (E1, "A244");
   end if;
when E2 : others => Unexpected_Exception (E2, "A245");
end;

------------------------------------------------------------------------
--  The Set and Get_Socket_Send_Buffer_Size methods are consistent
--  Doesn't work with a conenction mode sockete

Test ("Set/Get_Socket_Send_Buffer_Size [18.4.9]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count := 16;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Set_Socket_Send_Buffer_Size (Socket, 32);
   Comment ("Socket Send buffer size set to 32");
   To := Get_Socket_Send_Buffer_Size (Socket);
   Assert (To = 32, "A258");
exception when E : others => Unexpected_Exception (E, "A259");
end;

------------------------------------------------------------------------
--  The Get_Socket_Send_Buffer_Size method has a protocol defined
--  default value, wich is verly unlikly to be 0

Test ("Get_Socket_Send_Buffer_Size (Default) [18.4.9.2]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count := 0;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   To := Get_Socket_Send_Buffer_Size (Socket);
   Assert (To /= 0, "A260");
exception when E : others => Unexpected_Exception (E, "A261");
end;

------------------------------------------------------------------------
--  The Set_Socket_Send_Buffer_Size causes the Not_A_Socket
--  POSIX error to be raised when the socket has not been created

Test ("Set_Socket_Send_Buffer_Size: Not_A_Socket [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count           := 256;
begin
   Set_Socket_Send_Buffer_Size (Socket, To);
   Expect_Exception ("A262");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Not_A_Socket then
```

```
      Unexpected_Exception (E1, "A263");
   end if;
when E2 : others => Unexpected_Exception (E2, "A264");
end;

------------------------------------------------------------------------
--  The Get_Socket_Send_Buffer_Size causes the Not_A_Socket
--  POSIX error to be raised when the socket has not been created

Test ("Get_Socket_Send_Buffer_Size: Not_A_Socket [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count           := 256;
begin
   To := Get_Socket_Send_Buffer_Size (Socket);
   Expect_Exception ("A265");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Not_A_Socket then
      Unexpected_Exception (E1, "A266");
   end if;
when E2 : others => Unexpected_Exception (E2, "A267");
end;

------------------------------------------------------------------------
--  The Set_Socket_Send_Buffer_Size causes a No_Buffer_Space
--  Error when the requested buffer is too big

Test ("Set_Socket_Send_Buffer_Size: No_Buffer_Space [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
   To     : POSIX.IO_Count           := 10000000;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Set_Socket_Send_Buffer_Size (Socket, To);
   Expect_Exception ("A270");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= No_Buffer_Space then
      Unexpected_Exception (E1, "A271");
   end if;
when E2 : others => Unexpected_Exception (E2, "A272");
end;

------------------------------------------------------------------------
--  The Set and Get_Socket_Send_Low_Water_Mark methods are
--  consistent if the implementation allows the value to be set.
--  Otherwise the Permission_Denied Error will be raised
--  If a system does not even support the Send_Low_Water_Mark
--  then the Unknown_Protocol_Option error will be raised

Test ("Set/Get_Socket_Send_Low_Water_Mark [18.4.9]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count := 2;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Set_Socket_Send_Low_Water_Mark (Socket, 3);
   Comment ("Socket Send buffer size set to 3");
   To := Get_Socket_Send_Low_Water_Mark (Socket);
   Assert (To = 3, "A273");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code = Unknown_Protocol_Option then
      Comment ("Unknown_Protocol_Option");
   else
      Unexpected_Exception (E1, "A274");
   end if;
when E2 : others => Unexpected_Exception (E2, "A275");
end;

------------------------------------------------------------------------
--  The Get_Socket_Send_Low_Water_Mark method has a default value
--  of 1

Test ("Get_Socket_Send_Low_Water_Mark (Default) [18.4.9.2]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count := 0;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   To := Get_Socket_Send_Low_Water_Mark (Socket);
   Assert (To = 1, "A276");
exception
when POSIX_Error =>
   if Get_Error_Code = Unknown_Protocol_Option then
      Comment ("Unknown_Protocol_Option");
   end if;
when E2 : others => Unexpected_Exception (E2, "A277");
end;

------------------------------------------------------------------------
--  The Set_Socket_Send_Low_Water_Mark method causes the
--  Not_A_Socket POSIX error to be raised when the socket
--  has not been created (See above for other cases)

Test ("Set_Socket_Send_Low_Water_Mark: Not_A_Socket [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count           := 2;
begin
   Set_Socket_Send_Low_Water_Mark (Socket, To);
   Expect_Exception ("A278");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code /= Not_A_Socket then
      Unexpected_Exception (E1, "A279");
   end if;
when E2 : others => Unexpected_Exception (E2, "A280");
end;

------------------------------------------------------------------------
--  The Get_Socket_Send_Low_Water_Mark method causes the
--  Not_A_Socket POSIX error to be raised when the socket has not
--  been created (see above for other cases)

Test ("Get_Socket_Send_Low_Water_Mark: Not_A_Socket [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : POSIX.IO_Count           := 2;
begin
   To := Get_Socket_Send_Buffer_Size (Socket);
   Expect_Exception ("A281");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code /= Not_A_Socket then
      Unexpected_Exception (E1, "A282");
   end if;
when E2 : others => Unexpected_Exception (E2, "A283");
end;

------------------------------------------------------------------------
--  The Set and Get_Socket_Send_Timeout methods are consistent

Test ("Set/Get_Socket_Send_Timeout [18.4.9]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : Duration := 16.0;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Set_Socket_Send_Timeout (Socket, 32.0);
   Comment ("Socket Send timout set to 32");
   To := Get_Socket_Send_Timeout (Socket);
   Assert (To = 32.0, "A286");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code = Unknown_Protocol_Option then
      Comment ("Unknown_Protocol_Option");
   else
      Unexpected_Exception (E1, "A287");
   end if;
when E2 : others => Unexpected_Exception (E2, "A288");
end;

------------------------------------------------------------------------
--  The Get_Socket_Send_Timeout method has a defined
--  default value of 0

Test ("Get_Socket_Send_Timeout (Default) [18.4.9.2]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   To     : Duration                 := 2.0;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   To := Get_Socket_Send_Timeout (Socket);
   Assert (To = 0.0, "A289");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code = Unknown_Protocol_Option then
      Comment ("Unknown_Protocol_Option");
   else
      Unexpected_Exception (E1, "A290");
   end if;
when E2 : others => Unexpected_Exception (E2, "A291");
end;

------------------------------------------------------------------------
--  The Set_Socket_Send_Timeout causes the Not_A_Socket
--  POSIX error to be raised when the socket has not been created

Test ("Set_Socket_Send_Timout: Not_A_Socket [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 1;
   To     : Duration                 := 3.0;
begin
   Set_Socket_Send_Timeout (Socket, To);
   Expect_Exception ("A292");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code /= Not_A_Socket then
      Unexpected_Exception (E1, "A293");
   end if;
when E2 : others => Unexpected_Exception (E2, "A294");
end;

------------------------------------------------------------------------
--  The Get_Socket_Send_Timeout causes the Not_A_Socket
--  POSIX error to be raised when the socket has not been created
```

```
Test ("Get_Socket_Send_Timeout: Not_A_Socket [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 3;
   To     : Duration                 := 2.0;
begin
   To := Get_Socket_Send_Timeout (Socket);
   Expect_Exception ("A295");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code /= Not_A_Socket then
      Unexpected_Exception (E1, "A296");
   end if;
when E2 : others => Unexpected_Exception (E2, "A297");
end;

------------------------------------------------------------------------
--  The Set_Socket_Send_Timout causes a Domain_Error
--  Error when the requested buffer is too big

Test ("Set_Socket_Send_Timeout: Domain_Error [18.4.9.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
   To     : Duration := 10000000.0;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Set_Socket_Send_Timeout (Socket, To);
   Expect_Exception ("A300");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code = Permission_Denied then
      Comment ("Set not allowed -- Test not valid");
   elsif Get_Error_Code = Unknown_Protocol_Option then
      Comment ("Unknown_Protocol_Option");
   elsif Get_Error_Code /= No_Buffer_Space then
      Unexpected_Exception (E1, "A301");
   end if;
when E2 : others => Unexpected_Exception (E2, "A302");
end;

------------------------------------------------------------------------
--  The Is_A_Socket function returns a value of True if the input is
--  a socket and False otherwise

Test ("Is_A_Socket [18.4.10]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   Is_A   : Boolean;
begin
   Is_A := Is_A_Socket (Socket);
   Assert (Is_A = FALSE, "A303");
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Is_A := Is_A_Socket (Socket);
   Assert (Is_A = TRUE, "A304");
exception when E : others => Unexpected_Exception (E, "A305");
end;

------------------------------------------------------------------------
--  A closed socket will cause Is_A_Socket to raise the
--  Bad_File_Desriptor error code.

Test ("Is_A_Socket: Bad_File_Descriptor [18.4.10]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   Is_A   : Boolean;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Close (Socket);
   Is_A := Is_A_Socket (Socket);
   Expect_Exception ("A306");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Bad_File_Descriptor then
      Unexpected_Exception (E1, "A307");
   end if;
when E2 : others => Unexpected_Exception (E2, "A308");
end;

------------------------------------------------------------------------
--  The Set and Get_Socket_Broadcast method tests

declare
   procedure Test_Socket_Broadcast is new Testit
      (Socket_Option_Value, Get_Socket_Broadcast,
       Set_Socket_Broadcast);
begin
   Test_Socket_Broadcast ("Socket_Broadcast", Enabled, Disabled,
      "A126", "A127", "A128", "A129", "A130", "A131", "A132",
      "A133", "A134", "A135");
end;

------------------------------------------------------------------------
--  The Set and Get_Socket_Debugging method tests

declare
   procedure Test_Socket_Debugging is new Testit
      (Socket_Option_Value, Get_Socket_Debugging,
       Set_Socket_Debugging);
begin
   Test_Socket_Debugging ("Socket_Debugging", Enabled, Disabled,
      "A138", "A139", "A140", "A141", "A142", "A143", "A144",
      "A145", "A146", "A147");
end;

------------------------------------------------------------------------
--  The Set and Get_Socket_Routing method tests

declare
   procedure Test_Socket_Routing is new Testit
      (Socket_Option_Value, Get_Socket_Routing,
       Set_Socket_Routing);
begin
   Test_Socket_Routing ("Socket_Routing", Disabled, Enabled,
      "A150", "A151", "A152", "A153", "A154", "A155", "A156",
      "A157", "A158", "A159");
end;

------------------------------------------------------------------------
--  The Set and Get_Socket_Keep_Alive method tests

declare
   procedure Test_Socket_Keep_Alive is new Testit
      (Socket_Option_Value, Get_Socket_Keep_Alive,
       Set_Socket_Keep_Alive);
begin
   Test_Socket_Keep_Alive ("Socket_Keep_Alive", Enabled, Disabled,
      "A162", "A163", "A164", "A165", "A166", "A167", "A168",
      "A169", "A170", "A171");
end;

------------------------------------------------------------------------
--  The Set and Get_Socket_Linger_Time method tests

declare
   procedure Test_Socket_Linger_Time is new Testit
      (Linger_Time, Get_Socket_Linger_Time,
       Set_Socket_Linger_Time);
begin
   Test_Socket_Linger_Time ("Socket_Linger_Time", 2, 0,
      "A177", "A178", "A179", "A180", "A181", "A182", "A183",
      "A184", "A185", "A186");
end;

------------------------------------------------------------------------
--  The Set and Get_Socket_OOB_Data_Inline method tests

declare
   procedure Test_Socket_OOB_Data_Inline is new Testit
      (Socket_Option_Value, Get_Socket_OOB_Data_Inline,
       Set_Socket_OOB_Data_Inline);
begin
   Test_Socket_OOB_Data_Inline ("Socket_OOB_Data_Inline",
      Enabled, Disabled, "A189", "A190", "A191", "A192", "A193",
      "A194", "A195", "A196", "A197", "A198");
end;

------------------------------------------------------------------------
--  The Set and Get_Socket_Reuse_Addresses method tests

declare
   procedure Test_Socket_Reuse_Addresses is new Testit
      (Socket_Option_Value, Get_Socket_Reuse_Addresses,
       Set_Socket_Reuse_Addresses);
begin
   Test_Socket_Reuse_Addresses ("Reuse_Addresses",
      Enabled, Disabled, "A246", "A247", "A248", "A249", "A250",
      "A251", "A252", "A253", "A254", "A255");
end;

------------------------------------------------------------------------
--  A Socket can be bound

Test ("Bind [18.4.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Bind_Tests (Socket, Error1 => "", Error2 => "A001",
      Error3 => "A002");
   Close (Socket);
end;

------------------------------------------------------------------------
--  Trying to bind a socket to a port that is reserved results in the
--  Permission_Denied error code.

Test ("Bind: Permission_Denied [18.4.3.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Bind_Tests (Socket, Port => 23, Expected => Permission_Denied,
      Error1 => "A003", Error2 => "A004", Error3 => "A005");
   Close (Socket);
end;

------------------------------------------------------------------------
--  Trying to bind a socket to a port already in use results in the
--  Already_In_Use error code.

Test ("Bind: Already_In_Use [18.4.3.3]");
declare
   Socket1 : POSIX_IO.File_Descriptor;
   Socket2 : POSIX_IO.File_Descriptor;
begin
   Socket1 := Create (Internet_Protocol, Stream_Socket, TCP);
   Bind_Tests (Socket1, Port => 1530, Error1 => "",
      Error2 => "A006", Error3 => "A007");
```

```
   Socket2 := Create (Internet_Protocol, Stream_Socket, TCP);
   Bind_Tests (Socket2, Port => 1530, Expected => Address_In_Use,
      Error1 => "A008", Error2 => "A009", Error3 => "A010");
   Close (Socket1);
   Close (Socket2);
end;

------------------------------------------------------------------------
--  Trying to bind a socket to an address that doesn't exist results
--  in the Address_Not_Available error code.

Test ("Bind: Address_Not_Available [18.4.3.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Bind_Tests (Socket,
      Address => String_To_Internet_Address ("@#$%^&*()"),
      Expected => Address_Not_Available, Error1 => "A011",
      Error2 => "A012", Error3 => "A013");
   Close (Socket);
end;


------------------------------------------------------------------------
--  Trying to bind a closed socket results
--  in the Bad_File_Descriptor error code.

Test ("Bind: Bad_File_Descriptor [18.4.3.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Close (Socket);
   Bind_Tests (Socket, Expected => Bad_File_Descriptor,
      Error1 => "A014", Error2 => "A015", Error3 => "A016");
end;

------------------------------------------------------------------------
--  Trying to bind a non socket results
--  in the Not_A_Socket error code.

Test ("Bind: Not_A_Socket [18.4.3.3]");
declare
   Socket  : POSIX_IO.File_Descriptor := 0;
begin
   Bind_Tests (Socket, Expected => Not_A_Socket,
      Error1 => "A017", Error2 => "A018", Error3 => "A019");
end;

------------------------------------------------------------------------
--  Trying to bind to a socket with a different type of object results
--  in the Inappropriate_Family error code.  This error is not in the
--  the standard but I get it in what I think is the place of
--  Incorrect_Address_Type.

Test ("Bind: Inappropriate_Family [18.4.3.3]");
declare
   Socket  : POSIX_IO.File_Descriptor := 0;
begin
   Socket := Create (Inappro_Family, Stream_Socket, Dummy_Protocol);
   Bind_Tests (Socket, Expected => Inappropriate_Family,
      Error1 => "A020", Error2 => "A021", Error3 => "A022");
end;

------------------------------------------------------------------------
--  Trying to bind a socket with a bad Address pointer results
--  in the Invalid_Argument error code.

Test ("Bind: Invalid_Argument [18.4.3.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
   Name    : Internet_Socket_Address_Pointer;
   Junk    : Internet_Address;
   function To_Internet_Socket_Address_Pointer is new
      Unchecked_Conversion (System.Address,
         Internet_Socket_Address_Pointer);
begin
   Name := To_Internet_Socket_Address_Pointer (Junk'Address);
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Bind (Socket, +Name);
   Expect_Exception ("A023");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Invalid_Argument then
      Unexpected_Exception (E1, "A024");
   end if;
when E2 : others => Unexpected_Exception (E2, "A025");
end;

------------------------------------------------------------------------
--  Connecting a socket to local machines telnet port

Test ("Connect [18.4.4]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Connect_Tests (Socket, 23, String_To_Internet_Address ("127.0.0.1"),
      No_Error, "", "A026", "A027");
   Close (Socket);
end;

------------------------------------------------------------------------
--  Trying to bind a socket to a port that is reserved results in the
--  Permission_Denied error code.  Since connect choses the port this
--  error message cannot be systematicaly created.
------------------------------------------------------------------------

------------------------------------------------------------------------
--  Trying to bind a socket already in use results in the
--  Already_In_Use error code.  Since connect choses the port to bind
--  to this error message cannot by systematicaly created.
------------------------------------------------------------------------

------------------------------------------------------------------------
--  Trying to connect a socket already connected results in the
--  Is_Already_Connected error code.

Test ("Connect: Is_Already_Connected [18.4.4.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Connect_Tests (Socket, Error1 => "",
      Error2 => "A028", Error3 => "A029");
   Connect_Tests (Socket, Expected => Is_Already_Connected,
      Error1 => "A030", Error2 => "A031", Error3 => "A032");
   Close (Socket);
end;

------------------------------------------------------------------------
--  Trying to connect a socket to an address that doesn't exist results
--  in the Network_Unreachable error code.

Test ("Connect: Network_Unreachable [18.4.4.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Connect_Tests (Socket,
      Address => String_To_Internet_Address ("@#$%^&*()"),
      Expected => Network_Unreachable, Error1 => "A033",
      Error2 => "A034", Error3 => "A035");
   Close (Socket);
end;

------------------------------------------------------------------------
--  Address_Not_Available is an error code generated by the bind
--  portion of connect.  In all createable cases this error is
--  supercided by the Network_Unreachabe error code.
------------------------------------------------------------------------

------------------------------------------------------------------------
--  Trying to connect a closed socket results
--  in the Bad_File_Descriptor error code.

Test ("Connect: Bad_File_Descriptor [18.4.4.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Close (Socket);
   Connect_Tests (Socket, Expected => Bad_File_Descriptor,
      Error1 => "A036", Error2 => "A037", Error3 => "A038");
end;

------------------------------------------------------------------------
--  Trying to connect a non socket results
--  in the Not_A_Socket error code.

Test ("Connect: Not_A_Socket [18.4.3.3]");
declare
   Socket  : POSIX_IO.File_Descriptor := 0;
begin
   Connect_Tests (Socket, Expected => Not_A_Socket,
      Error1 => "A039", Error2 => "A040", Error3 => "A041");
end;

------------------------------------------------------------------------
--  Trying to connect to a socket with a different type of object
--  results in the Inappropriate_Family error code.  This error is not
--  in the the standard but I get it in what I think is the place of
--  Incorrect_Address_Type.

Test ("Connect: Inappropriate_Family [18.4.4.3]");
declare
   Socket  : POSIX_IO.File_Descriptor := 0;
begin
   Socket := Create (Inappro_Family, Stream_Socket, Dummy_Protocol);
   Connect_Tests (Socket, Expected => Inappropriate_Family,
      Error1 => "A042", Error2 => "A043", Error3 => "A044");
end;

------------------------------------------------------------------------
--  Trying to connect with a bad Address pointer results
--  in the Invalid_Argument error code.

Test ("Connect: Invalid_Argument [18.4.4.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
   Name    : Internet_Socket_Address_Pointer;
   Junk    : Internet_Address;
   function To_Internet_Socket_Address_Pointer is new
      Unchecked_Conversion (System.Address,
         Internet_Socket_Address_Pointer);
begin
   Name := To_Internet_Socket_Address_Pointer (Junk'Address);
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Connect (Socket, +Name);
```

```
   Expect_Exception ("A045");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Invalid_Argument then
      Unexpected_Exception (E1, "A046");
   end if;
when E2 : others => Unexpected_Exception (E2, "A047");
end;

-----------------------------------------------------------------------
--  Trying to connect twice to a non-reponding address causes the
--  Operation_In_Progress error code.

Test ("Connect: Operation_In_Progress [18.4.4.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
   Int_Add : aliased Internet_Socket_Address;
   Name    : Internet_Socket_Address_Pointer := Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, 23);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address (Time_Out_URL));
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Set_File_Control (Socket, Non_Blocking);
   Connect (Socket, +Name);
   Connect (Socket, +Name);
   Expect_Exception ("A048");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Operation_In_Progress then
      Unexpected_Exception (E1, "A049");
   end if;
when E2 : others => Unexpected_Exception (E2, "A050");
end;

-----------------------------------------------------------------------
--  The Operation_In_Progress error code seem to supercede the
--  Already_Awaiting_Connection error code.  Therefore there is no
--  way to create an Alredy_Awaiting_Connection error code.
-----------------------------------------------------------------------

-----------------------------------------------------------------------
--  Trying to connect to a restricted port causes
--  the Connection_Refused error code.

Test ("Connect: Connection_Refused [18.4.4.3]");
declare
   Socket  : POSIX_IO.File_Descriptor := 0;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Connect_Tests (Socket, Port => Restricted_Port,
      Expected => Connection_Refused, Error1 => "A051",
      Error2 => "A052", Error3 => "A053");
end;

-----------------------------------------------------------------------
--  ??? Host_Unreachable and Network_Unreachable appear to be the
--  ??? same thing.  Either that or connect would simply block.
-----------------------------------------------------------------------

-----------------------------------------------------------------------
--  Any signal that would cause Interrupted_Argument on a single
--  threaded process will stop the process.  Therefore this case
--  cannot be recreated.
-----------------------------------------------------------------------

-----------------------------------------------------------------------
--  The Network_Down error code cannot accuratly be created due to
--  it the unreliability of a network being down.
-----------------------------------------------------------------------

-----------------------------------------------------------------------
--  The No_Buffer_Space error code cannot be accuratly created due to
--  the need of buffer space for the rest of the test.
-----------------------------------------------------------------------

-----------------------------------------------------------------------
--  Trying to connect to a non-reponding address causes the
--  Timed_Out error code after a period of time.

Test ("Connect: Timed_Out (This may take a few minutes) [18.4.4.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Connect_Tests (Socket, Address =>
      String_To_Internet_Address (Time_Out_URL),
      Expected => Timed_Out, Error1 => "A054", Error2 => "A055",
      Error3 => "A056");
end;

-----------------------------------------------------------------------
--  Specifying Peer as local machines telnet port

Test ("Specify_Peer [18.4.4]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Specify_Peer_Tests (Socket, 23, String_To_Internet_Address ("127.0.0.1"),
      No_Error, "", "A057", "A058");
   Close (Socket);
end;

-----------------------------------------------------------------------
--  Trying to specify a peer using a closed socket results
--  in the Bad_File_Descriptor error code.

Test ("Specify_Peer: Bad_File_Descriptor [18.4.4.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Close (Socket);
   Specify_Peer_Tests (Socket, Expected => Bad_File_Descriptor,
      Error1 => "A059", Error2 => "A060", Error3 => "A061");
end;

-----------------------------------------------------------------------
--  Trying to specify a peer using a non socket results
--  in the Not_A_Socket error code.

Test ("Specify_Peer: Not_A_Socket [18.4.3.3]");
declare
   Socket  : POSIX_IO.File_Descriptor := 0;
begin
   Specify_Peer_Tests (Socket, Expected => Not_A_Socket,
      Error1 => "A062", Error2 => "A063", Error3 => "A064");
end;

-----------------------------------------------------------------------
--  Trying to specify a peer useing a socket with a different type of
--  object results in the Inappropriate_Family error code.  This error
--  is not in the the standard but I get it in what I think is the
--  place of Incorrect_Address_Type.

Test ("Specify_Peer: Inappropriate_Family [18.4.4.3]");
declare
   Socket  : POSIX_IO.File_Descriptor := 0;
begin
   Socket := Create (Inappro_Family, Stream_Socket, Dummy_Protocol);
   Specify_Peer_Tests (Socket, Expected => Inappropriate_Family,
      Error1 => "A065", Error2 => "A066", Error3 => "A067");
end;

-----------------------------------------------------------------------
--  Trying to specify a peer with a bad Address pointer results
--  in the Invalid_Argument error code.

Test ("Specify_Peer: Invalid_Argument [18.4.4.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
   Name    : Internet_Socket_Address_Pointer;
   Junk    : Internet_Address;
   function To_Internet_Socket_Address_Pointer is new
      Unchecked_Conversion (System.Address,
         Internet_Socket_Address_Pointer);
begin
   Name := To_Internet_Socket_Address_Pointer (Junk'Address);
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Specify_Peer (Socket, +Name);
   Expect_Exception ("A068");
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Invalid_Argument then
      Unexpected_Exception (E1, "A069");
   end if;
when E2 : others => Unexpected_Exception (E2, "A070");
end;

-----------------------------------------------------------------------
--  Unspecify a peer that is the local machines telnet port

Test ("Unpecify_Peer [18.4.4]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Specify_Peer_Tests (Socket, 23, String_To_Internet_Address ("127.0.0.1"),
      No_Error, "", "A071", "A072");
   Unspecify_Peer_Tests (Socket, Error1 => "", Error2 => "A073",
      Error3 => "A074");
   Close (Socket);
end;

-----------------------------------------------------------------------
--  Unspecify a peer that hasn't been specified give the
--  Invalid_Argument error code.

Test ("Unspecify_Peer: Invalid_Argument [18.4.4]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Unspecify_Peer_Tests (Socket, Expected => Invalid_Argument,
      Error1 => "", Error2 => "A075", Error3 => "A076");
   Close (Socket);
end;

-----------------------------------------------------------------------
--  Set socket to listen on system chosen port

Test ("Listen [18.4.11]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Listen_Tests (Socket, Error1 => "",
      Error2 =>  "A077", Error3 => "A078");
   Close (Socket);
```

```
end;

------------------------------------------------------------------------
--  Trying to listen on a closes socket results in the
--  Bad_File_Desriptor error code

Test ("Listen: Bad_File_Desriptor [18.4.11.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Listen_Tests (Socket, Expected => Bad_File_Descriptor,
      Error1 => "A079", Error2 =>  "A080", Error3 => "A081",
      Close_Socket => true);
end;

------------------------------------------------------------------------
--  Trying to listen on a non-socket results in the
--  Not_A_Socket error code

Test ("Listen: Not_A_Socket [18.4.11.3]");
declare
   Socket  : POSIX_IO.File_Descriptor := 0;
begin
   Listen_Tests (Socket, Expected => Not_A_Socket,
      Error1 => "A082", Error2 =>  "A083", Error3 => "A084",
      Close_Socket => true);
end;

------------------------------------------------------------------------
--  Trying to listen on a UDP socket results in the
--  Option_Not_Supported error code

Test ("Listen: Option_Not_Supported [18.4.11.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Listen_Tests (Socket, Expected => Option_Not_Supported,
      Error1 => "A085", Error2 =>  "A086", Error3 => "A087");
end;

------------------------------------------------------------------------
--  Shutdown a connection

Test ("Shutdown [18.4.14]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Shutdown_Tests (Socket,
      Mode => Further_Sends_And_Receives_Disallowed,
      Error1 => "", Error2 =>  "A088", Error3 => "A089");
   Close (Socket);
end;

------------------------------------------------------------------------
--  Trying to shutdown a closed socket results in the
--  Bad_File_Desriptor error code.

Test ("Shutdown: Bad_File_Descriptor [18.4.14.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Shutdown_Tests (Socket, Expected => Bad_File_Descriptor,
      Mode => Further_Sends_And_Receives_Disallowed,
      Close_Socket => true, Error1 => "A090", Error2 =>  "A091",
      Error3 => "A092");
end;

------------------------------------------------------------------------
--  Trying to shutdown using a non-connected socket results in the
--  Not_Connected error code.

Test ("Shutdown: Not_Connected [18.4.14.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Shutdown_Tests (Socket, Mode => Further_Sends_Disallowed,
      Error1 => "A093", Error2 =>  "A094", Error3 => "A095",
      Expected => Not_Connected, Should_Connect => false);
end;

------------------------------------------------------------------------
--  Trying to shutdown using a non-socket results in the
--  Not_A_Socket error code.

Test ("Shutdown: Not_A_Socket [18.4.14.3]");
declare
   Socket  : POSIX_IO.File_Descriptor := 0;
begin
   Shutdown_Tests (Socket, Mode => Further_Sends_Disallowed,
      Error1 => "A096", Error2 =>  "A097", Error3 => "A098",
      Expected => Not_A_Socket, Should_Connect => false);
end;


------------------------------------------------------------------------
--                                                                    --
--  The few remaining functions require multiple processes to be      --
--  used to test them.  Please refer to the p180402 test files.       --
--                                                                    --
------------------------------------------------------------------------

   Done;
exception when E : others => Fatal_Exception (E, "A999");
end p180401;
```

p180402.adb

```
------------------------------------------------------------------------------
--                                                                          --
--                       POSIX.5c VALIDATION TEST SUITE                     --
--                                                                          --
--                              P 1 8 0 4 0 2                               --
--                                                                          --
--                                 B o d y                                  --
--                                                                          --
--                                                                          --
--  Copyright (c) 1995-1999 Florida  State  University  (FSU).  All Rights  --
--  Reserved.                                                               --
--                                                                          --
--  This is free software;  you can redistribute it and/or modify it under  --
--  terms of the  GNU  General  Public  License  as published by the  Free  --
--  Software Foundation;  either version 2, or (at your option) any  later  --
--  version.  This  software  is distributed  in the hope that it  will be  --
--  useful, but WITHOUT ANY WARRANTY; without even the implied warranty of  --
--  MERCHANTABILITY   or  FITNESS FOR A PARTICULAR PURPOSE.   See the  GNU  --
--  General Public License for more details.  You  should have  received a  --
--  copy of the GNU General Public License  distributed  with  GNARL;  see  --
--  file  COPYING.  If not,  write to  the  Free  Software  Foundation, 59  --
--  Temple Place - Suite 330, Boston, MA 02111-1307, USA.                   --
--                                                                          --
--  Under contract  GS-35F-4506G, the U. S. Government obtained  unlimited  --
--  rights in the software and documentation contained herein.   Unlimited  --
--  rights are defined in DFAR 252,227-7013(a)(19).  By making this public  --
--  release,   the  Government   intends   to   confer  upon  all  recipients  --
--  unlimited  rights equal to those held by the Government.  These rights  --
--  include rights to use,  duplicate,  release  or  disclose the released  --
--  data an computer software  in whole or in part,  in any manner and for  --
--  any purpose whatsoever, and to have or permit others to do so.          --
--                                                                          --
--  DISCLAIMER   --   ALL MATERIALS OR INFORMATION HEREIN RELEASED,   MADE  --
--  AVAILABLE OR DISCLOSED ARE AS IS.   THE GOVERNMENT MAKES NO EXPRESS OR  --
--  IMPLIED WARRANTY AS TO ANY MATTER WHATSOEVER, INCLUDING THE CONDITIONS  --
--  OF THE SOFTWARE,  DOCUMENTATION  OR  OTHER INFORMATION RELEASED,  MADE  --
--  AVAILABLE OR DISCLOSED,  OR THE OWNERSHIP, MERCHANTABILITY, OR FITNESS  --
--  FOR A PARTICULAR PURPOSE OF SAID MATERIAL.                              --
--                                                                          --
------------------------------------------------------------------------------

--  Multiprocess integrated test for package POSIX_Sockets
--  in IEEE Std 1003.5c Section 18.4 with package
--  POSIX_Sockets_Interent.

--  This test covers only features that depend only on
--  the packages (POSIX_Sockets/_Interent) and features from
--  other packages that are required to be supported.

with POSIX,
     POSIX_Sockets,
     POSIX_Sockets_Internet,
     POSIX_Report,
     POSIX_IO,
     POSIX_Process_Primitives,
     System,
     POSIX_Process_Identification;

procedure p180402 is

   use POSIX,
       POSIX_Sockets,
       POSIX_Sockets_Internet,
       POSIX_IO,
       POSIX_Process_Primitives,
       POSIX_Process_Identification,
       POSIX_Report;

   ChildA_Pathname : POSIX_String := "./p180402a";
   ChildA_Name     : POSIX_String := "p180402a";
   ChildB_Pathname : POSIX_String := "./p180402b";
   ChildB_Name     : POSIX_String := "p180402b";
   ChildC_Pathname : POSIX_String := "./p180402c";
   ChildC_Name     : POSIX_String := "p180402c";
   ChildD_Pathname : POSIX_String := "./p180402d";
   ChildD_Name     : POSIX_String := "p180402d";
   ChildE_Pathname : POSIX_String := "./p180402e";
   ChildE_Name     : POSIX_String := "p180402e";
   ChildF_Pathname : POSIX_String := "./p180402f";
   ChildF_Name     : POSIX_String := "p180402f";
   ChildG_Pathname : POSIX_String := "./p180402g";
   ChildG_Name     : POSIX_String := "p180402g";

   procedure Accept_Connection_Procedure (
      Socket       : in POSIX_IO.File_Descriptor;
      Address      : in Socket_Address_Pointer
                   := Null_Socket_Address;
      Expected     : in Error_Code := No_Error;
      Er1, Er2, Er3, Er4 : in String);

   procedure Accept_Connection_Function (
      Socket       : in POSIX_IO.File_Descriptor;
      Expected     : in Error_Code := No_Error;
      Er1, Er2, Er3, Er4 : in String);
```

```
procedure Receive1_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Options  : in Message_Option_Set := Empty_Set;
   Expected : in Error_Code := No_Error;
   Er1, Er2, Er3, Er4 : in String);

procedure Receive2_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Options  : in Message_Option_Set := Empty_Set;
   Expected : in Error_Code := No_Error;
   Er1, Er2, Er3, Er4 : in String);

procedure Receive3_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Options  : in Message_Option_Set := Empty_Set;
   From     : in Socket_Address_Pointer
            := Null_Socket_Address;
   Expected : in Error_Code := No_Error;
   Er1, Er2, Er3, Er4 : in String);

procedure Receive4_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Options  : in Message_Option_Set := Empty_Set;
   From     : in Socket_Address_Pointer
            := Null_Socket_Address;
   Expected : in Error_Code := No_Error;
   Er1, Er2, Er3, Er4 : in String);

procedure Receive_Message1_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Message  : in out Socket_Message;
   Options  : in Message_Option_Set := Empty_Set;
   Expected : in Error_Code := No_Error;
   Er1, Er2, Er3, Er4 : in String;
   Received_Length : out POSIX.IO_Count);

procedure Receive_Message2_Tests (
   Socket          : in POSIX_IO.File_Descriptor;
   Message         : in out Socket_Message;
   Options         : in Message_Option_Set := Empty_Set;
   Expected        : in Error_Code := No_Error;
   Er1, Er2, Er3, Er4 : in String;
   Received_Length : out POSIX.IO_Count);

procedure Accept_Connection_Procedure (
   Socket   : in POSIX_IO.File_Descriptor;
   Address  : in Socket_Address_Pointer
            := Null_Socket_Address;
   Expected : in Error_Code := No_Error;
   Er1, Er2, Er3, Er4 : in String)
is
   Connection_Socket : POSIX_IO.File_Descriptor := 0;
   Int_Add           : aliased Internet_Socket_Address;
   Connect_Name      : Socket_Address_Pointer :=
      +(Int_Add'Unchecked_Access);
begin
   if Address /= Null_Socket_Address then
      Connect_Name := Address;
   end if;
   Accept_Connection (Socket, Connection_Socket, Connect_Name);
   if Er1 /= "" then
      Expect_Exception (Er1);
   else
      Assert (Connection_Socket /= 0, Er2);
   end if;
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Expected then
      Unexpected_Exception (E1, Er3);
   end if;
when E2 : others => Unexpected_Exception (E2, Er4);
end Accept_Connection_Procedure;

procedure Accept_Connection_Function (
   Socket   : in POSIX_IO.File_Descriptor;
   Expected : in Error_Code := No_Error;
   Er1, Er2, Er3, Er4 : in String)
is
   Connection_Socket : POSIX_IO.File_Descriptor := 0;
begin
   Connection_Socket := Accept_Connection (Socket);
   if Er1 /= "" then
      Expect_Exception (Er1);
   else
      Assert (Connection_Socket /= 0, Er2);
   end if;
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Expected then
      Unexpected_Exception (E1, Er3);
   end if;
when E2 : others => Unexpected_Exception (E2, Er4);
end Accept_Connection_Function;

procedure Receive1_Tests (
   Socket    : in POSIX_IO.File_Descriptor;
   Options   : in Message_Option_Set := Empty_Set;
   Expected  : in Error_Code := No_Error;
   Er1, Er2, Er3, Er4 : in String)
is
   Octets    : POSIX_String (1 .. 40);
   Buffer    : System.Address := Octets'Address;
   Requested : POSIX.IO_Count := 40;
   Received  : POSIX.IO_Count := 0;
   Mask      : POSIX.Signal_Masking := All_Signals;
begin
   Comment ("Receive 1");
   Receive (Socket, Buffer, Requested, Received, Mask, Options);
   Comment ("received octets = " &
      Integer'Image (Integer (Received)));
   if Er1 = "" then
      Assert (Received = 11 and Octets (1 .. 11) = "Send 1 test", Er2);
   else
      Expect_Exception (Er1);
   end if;
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Expected then
      Unexpected_Exception (E1, Er3);
   end if;
when E2 : others => Unexpected_Exception (E2, Er4);
end Receive1_Tests;

procedure Receive2_Tests (
   Socket    : in POSIX_IO.File_Descriptor;
   Options   : in Message_Option_Set := Empty_Set;
   Expected  : in Error_Code := No_Error;
   Er1, Er2, Er3, Er4 : in String)
is
   Octets    : POSIX_String (1 .. 40);
   Buffer    : System.Address := Octets'Address;
   Requested : POSIX.IO_Count := 40;
   Received  : POSIX.IO_Count := 0;
begin
   Comment ("Receive 2");
   Receive (Socket, Buffer, Requested, Received, Options);
   Comment ("received octets = " &
      Integer'Image (Integer (Received)));
   if Er1 = "" then
      Assert (Received = 11 and Octets (1 .. 11) = "Send 2 test", Er2);
   else
      Expect_Exception (Er1);
   end if;
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Expected then
      Unexpected_Exception (E1, Er3);
   end if;
when E2 : others => Unexpected_Exception (E2, Er4);
end Receive2_Tests;

procedure Receive3_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Options  : in Message_Option_Set := Empty_Set;
   From     : in Socket_Address_Pointer
            := Null_Socket_Address;
   Expected : in Error_Code := No_Error;
   Er1, Er2, Er3, Er4 : in String)
is
   Octets    : POSIX_String (1 .. 40);
   Buffer    : System.Address := Octets'Address;
   Requested : POSIX.IO_Count := 40;
   Received  : POSIX.IO_Count := 0;
   Mask      : POSIX.Signal_Masking := All_Signals;
begin
   Comment ("Receive 3");
   Receive (Socket, Buffer, Requested, Received, From,
      Mask, Options);
   Comment ("received octets = " &
      Integer'Image (Integer (Received)));
   if Er1 = "" then
      Assert (Received = 11 and Octets (1 .. 11) = "Send 3 test", Er2);
   else
      Expect_Exception (Er1);
   end if;
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Expected then
      Unexpected_Exception (E1, Er3);
   end if;
when E2 : others => Unexpected_Exception (E2, Er4);
end Receive3_Tests;

procedure Receive4_Tests (
   Socket   : in POSIX_IO.File_Descriptor;
   Options  : in Message_Option_Set := Empty_Set;
   From     : in Socket_Address_Pointer
            := Null_Socket_Address;
   Expected : in Error_Code := No_Error;
   Er1, Er2, Er3, Er4 : in String)
is
   Octets    : POSIX_String (1 .. 40);
   Buffer    : System.Address := Octets'Address;
   Requested : POSIX.IO_Count := 40;
   Received  : POSIX.IO_Count := 0;
begin
   Comment ("Receive 4");
   Receive (Socket, Buffer, Requested, Received, From,
      Options);
   Comment ("received octets = " &
      Integer'Image (Integer (Received)));
   if Er1 = "" then
      Assert (Received = 11 and Octets (1 .. 11) = "Send 4 test", Er2);
   else
      Expect_Exception (Er1);
   end if;
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Expected then
```

```
         Unexpected_Exception (E1, Er3);
      end if;
   when E2 : others => Unexpected_Exception (E2, Er4);
   end Receive4_Tests;

   procedure Receive_Message1_Tests (
      Socket             : in POSIX_IO.File_Descriptor;
      Message            : in out Socket_Message;
      Options            : in Message_Option_Set := Empty_Set;
      Expected           : in Error_Code := No_Error;
      Er1, Er2, Er3, Er4 : in String;
      Received_Length    : out POSIX.IO_Count)
   is
      Mask : POSIX.Signal_Masking := All_Signals;
   begin
      Comment ("Receive Message 1");
      Receive_Message (Socket, Message, Received_Length, Mask, Options);
      Comment ("received octets = " &
         Integer'Image (Integer (Received_Length)));
      if Er1 = "" then
         Assert (Received_Length /= 0, Er2);
      else
         Expect_Exception (Er1);
      end if;
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Expected then
         Unexpected_Exception (E1, Er3);
      end if;
   when E2 : others => Unexpected_Exception (E2, Er4);
   end Receive_Message1_Tests;

   procedure Receive_Message2_Tests (
      Socket             : in POSIX_IO.File_Descriptor;
      Message            : in out Socket_Message;
      Options            : in Message_Option_Set := Empty_Set;
      Expected           : in Error_Code := No_Error;
      Er1, Er2, Er3, Er4 : in String;
      Received_Length    : out POSIX.IO_Count)

   is
   begin
      Comment ("Receive Message 1");
      Receive_Message (Socket, Message, Received_Length, Options);
      Comment ("received octets = " &
         Integer'Image (Integer (Received_Length)));
      if Er1 = "" then
         Assert (Received_Length /= 0, Er2);
      else
         Expect_Exception (Er1);
      end if;
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Expected then
         Unexpected_Exception (E1, Er3);
      end if;
   when E2 : others => Unexpected_Exception (E2, Er4);
   end Receive_Message2_Tests;


------------------------------------------------------------------------
--  Begin Tests

begin
   Header ("p180402");
   Test ("package POSIX.Sockets");

   ---------------------------------------------------------------------
   --  Accept an incoming connection

   Test ("Accept_Connection Procedure [18.4.2]");
   declare
      Socket   : POSIX_IO.File_Descriptor;
      Port     : Positive;
      Template : Process_Template;
      Child_ID : Process_ID;
      Args     : POSIX_String_List;
      Int_Add  : aliased Internet_Socket_Address;
      Name     : Internet_Socket_Address_Pointer :=
         Int_Add'Unchecked_Access;
   begin
      Set_Internet_Port (Int_Add, Unspecified_Internet_Port);
      Set_Internet_Address (Int_Add,
         String_To_Internet_Address ("127.0.0.1"));
      Socket := Create (Internet_Protocol, Stream_Socket, TCP);
      Bind (Socket, +Name);
      Port := Integer (Get_Internet_Port (Get_Socket_Name (Socket)));
      Listen (Socket);

      Comment ("port is " & Integer'Image (Port));
      Open_Template (Template);
      Make_Empty (Args);
      POSIX.Append (Args, ChildA_Name);
      POSIX.Append (Args, "-child");
      POSIX.Append (Args, To_POSIX_String (Integer'Image
         (Port)));
      Pass_Through_Verbosity (Args);
      Comment ("Starting ChildA process");
      Start_Process (Child_ID, ChildA_Pathname, Template, Args);
      Close_Template (Template);
      Accept_Connection_Procedure (Socket, Er1 => "",
         Er2 =>  "A001", Er3 => "A002", Er4 => "A003");
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A004");
      Close (Socket);
   end;

   ---------------------------------------------------------------------
   --  An address object of an incorrect type for the address format of
   --  this socket raises the Incorrect_Address_Type error code.

   --  ... The EFAULT (Bad_Address) error was raised instead of the
   --  ... the Incorrect_Address_Type.  It is not clear whether the
   --  ... error was simply left out of the Ada standard or if the
   --  ... original c POSIX standard was not finalized when the Ada
   --  ... standard was written.

   ---------------------------------------------------------------------
   --  Trying to accept a connection on a closed socket results in the
   --  Bad_File_Descriptor error code.

   Test ("Accept_Connection Procedure: Bad_File_Descriptor [18.4.2.3]");
   declare
      Socket  : POSIX_IO.File_Descriptor;
   begin
      Socket := Create (Internet_Protocol, Stream_Socket, TCP);
      Close (Socket);
      Accept_Connection_Procedure (Socket,
         Expected => Bad_File_Descriptor,
         Er1 => "A009", Er2 =>  "A0010", Er3 => "A011", Er4 => "A012");
   end;

   ---------------------------------------------------------------------
   --  Trying to accept a connection from a non listening socket
   --  raises the Invalid_Argument error code.

   Test ("Accept_Connection Procedure: Invalid_Argument [18.4.2.3]");
   declare
      Socket  : POSIX_IO.File_Descriptor;
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
         Int_Add'Unchecked_Access;
   begin
      Set_Internet_Port (Int_Add, Unspecified_Internet_Port);
      Set_Internet_Address (Int_Add,
         String_To_Internet_Address ("127.0.0.1"));
      Socket := Create (Internet_Protocol, Stream_Socket, TCP);
      Bind (Socket, +Name);
      Accept_Connection_Procedure (Socket,
         Expected => Invalid_Argument,
         Er1 => "A013", Er2 =>  "A014", Er3 => "A015", Er4 => "A016");
      Close (Socket);
   end;

   ---------------------------------------------------------------------
   --  Trying to accept a connection from a non-socket
   --  raises the Not_A_Socket error code.

   Test ("Accept_Connection Procedure: Not_A_Socket [18.4.2.3]");
   declare
      Socket  : POSIX_IO.File_Descriptor := 0;
   begin
      Accept_Connection_Procedure (Socket,
         Expected => Not_A_Socket,
         Er1 => "A017", Er2 =>  "A018", Er3 => "A019", Er4 => "A020");
   end;

   ---------------------------------------------------------------------
   --  Trying to accept a connection from an socket that does not support
   --  the Accept_Connection operation raises the
   --  Option_Not_Supported error code.

   Test ("Accept_Connection Procedure: Option_Not_Supported [18.4.2.3]");
   declare
      Socket  : POSIX_IO.File_Descriptor;
   begin
      Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
      Accept_Connection_Procedure (Socket,
         Expected => Option_Not_Supported,
         Er1 => "A021", Er2 =>  "A022", Er3 => "A023", Er4 => "A024");
      Close (Socket);
   end;

   ---------------------------------------------------------------------
   --  A non-blocking socket will cause the Would_Block error code
   --  when Accept_Connection would normaly block.

   Test ("Accept_Connection Procedure: Would_Block [18.4.2.3]");
   declare
      Socket   : POSIX_IO.File_Descriptor;
      Int_Add  : aliased Internet_Socket_Address;
      Name     : Internet_Socket_Address_Pointer :=
         Int_Add'Unchecked_Access;
   begin
      Set_Internet_Port (Int_Add, Unspecified_Internet_Port);
      Set_Internet_Address (Int_Add,
         String_To_Internet_Address ("127.0.0.1"));
      Socket := Create (Internet_Protocol, Stream_Socket, TCP);
      Set_File_Control (Socket, Non_Blocking);
      Bind (Socket, +Name);
      Listen (Socket);
      Accept_Connection_Procedure (Socket,
         Expected => Would_Block,
         Er1 => "A025", Er2 =>  "A026",
         Er3 => "A027", Er4 => "A028");
      Close (Socket);
   end;

   ---------------------------------------------------------------------
```

```
--  Accept an incoming connection

Test ("Accept_Connection Function [18.4.2]");
declare
   Socket   : POSIX_IO.File_Descriptor;
   Port     : Positive;
   Template : Process_Template;
   Child_ID : Process_ID;
   Args     : POSIX_String_List;
   Int_Add  : aliased Internet_Socket_Address;
   Name     : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, Unspecified_Internet_Port);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Bind (Socket, +Name);
   Port := Integer (Get_Internet_Port (Get_Socket_Name (Socket)));
   Listen (Socket);

   Comment ("port is " & Integer'Image (Port));
   Open_Template (Template);
   Make_Empty (Args);
   POSIX.Append (Args, ChildA_Name);
   POSIX.Append (Args, "-child");
   POSIX.Append (Args, To_POSIX_String (Integer'Image
      (Port)));
   Pass_Through_Verbosity (Args);
   Comment ("Starting ChildA process");
   Start_Process (Child_ID, ChildA_Pathname, Template, Args);
   Close_Template (Template);
   Accept_Connection_Function (Socket, Er1 => "",
      Er2 =>  "A029", Er3 => "A030", Er4 => "A031");
   Close (Socket);
end;

-----------------------------------------------------------------------
--  Trying to accept a connection on a closed socket results in the
--  Bad_File_Descriptor error code.

Test ("Accept_Connection Function: Bad_File_Descriptor [18.4.2.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Close (Socket);
   Accept_Connection_Function (Socket,
      Expected => Bad_File_Descriptor,
      Er1 => "A032", Er2 =>  "A033", Er3 => "A034", Er4 => "A035");
end;

-----------------------------------------------------------------------
--  Trying to accept a connection from a non listening socket
--  raises the Invalid_Argument error code.

Test ("Accept_Connection Function: Invalid_Argument [18.4.2.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
   Int_Add : aliased Internet_Socket_Address;
   Name    : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, Unspecified_Internet_Port);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Bind (Socket, +Name);
   Accept_Connection_Function (Socket,
      Expected => Invalid_Argument,
      Er1 => "A036", Er2 =>  "A037", Er3 => "A038", Er4 => "A039");
   Close (Socket);
end;

-----------------------------------------------------------------------
--  Trying to accept a connection from a non-socket
--  raises the Not_A_Socket error code.

Test ("Accept_Connection Function: Not_A_Socket [18.4.2.3]");
declare
   Socket  : POSIX_IO.File_Descriptor := 0;
begin
   Accept_Connection_Function (Socket,
      Expected => Not_A_Socket,
      Er1 => "A040", Er2 =>  "A041", Er3 => "A042", Er4 => "A043");
end;

-----------------------------------------------------------------------
--  Trying to accept a connection from an socket that does not support
--  the Accept_Connection operation raises the
--  Option_Not_Supported error code.

Test ("Accept_Connection Function: Option_Not_Supported [18.4.2.3]");
declare
   Socket  : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Accept_Connection_Function (Socket,
      Expected => Option_Not_Supported,
      Er1 => "A044", Er2 =>  "A045", Er3 => "A046", Er4 => "A047");
   Close (Socket);
end;

-----------------------------------------------------------------------
--  A non-blocking socket will cause the Would_Block error code
--  when Accept_Connection would normaly block.

Test ("Accept_Connection Function: Would_Block [18.4.2.3]");
declare
   Socket   : POSIX_IO.File_Descriptor;
   Int_Add  : aliased Internet_Socket_Address;
   Name     : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, Unspecified_Internet_Port);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Set_File_Control (Socket, Non_Blocking);
   Bind (Socket, +Name);
   Listen (Socket);
   Accept_Connection_Function (Socket,
      Expected => Would_Block,
      Er1 => "A048", Er2 =>  "A049",
      Er3 => "A050", Er4 => "A051");
   Close (Socket);
end;

-----------------------------------------------------------------------
--                                                                   --
--  The remaining Accept_Connection errors cannot be tested due to  --
--  their dependence on either system resources or race conditions.  --
--                                                                   --
-----------------------------------------------------------------------

-----------------------------------------------------------------------
--  Receive a message sent to local socket

Test ("Receive 1 [18.4.12]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Port      : Positive;
   Template  : Process_Template;
   Child_ID  : Process_ID;
   Args      : POSIX_String_List;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
   Connection_Socket : POSIX_IO.File_Descriptor := 0;
begin
   Set_Internet_Port (Int_Add, Unspecified_Internet_Port);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Bind (Socket, +Name);
   Port := Integer (Get_Internet_Port (Get_Socket_Name (Socket)));
   Listen (Socket);

   Comment ("port is " & Integer'Image (Port));
   Make_Empty (Args);
   POSIX.Append (Args, ChildB_Name);
   POSIX.Append (Args, "-child");
   POSIX.Append (Args, To_POSIX_String (Integer'Image
      (Port)));
   Pass_Through_Verbosity (Args);
   Open_Template (Template);
   Comment ("Starting ChildB process");
   Start_Process (Child_ID, ChildB_Pathname, Template, Args);
   Close_Template (Template);
   Connection_Socket := Accept_Connection (Socket);
   Receive1_Tests (Connection_Socket, Er1 => "",
      Er2 => "A052", Er3 => "A053", Er4 => "A054");
   Close (Socket);
end;

-----------------------------------------------------------------------
--  A non-blocking socket that should block on a Receive generates
--  the Would_Block error code

Test ("Receive 1: Would_Block [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, 9);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Set_File_Control (Socket, Non_Blocking);
   Connect (Socket, +Name);
   Receive1_Tests (Socket, Expected => Would_Block,
      Er1 => "A055", Er2 => "A056",
      Er3 => "A057", Er4 => "A058");
   Close (Socket);
end;

-----------------------------------------------------------------------
--  Trying to receive on a closed file descriptor generates
--  the Bad_File_Desriptor error code.

Test ("Receive 1: Bad_File_Destriptor [18.4.12.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Close (Socket);
   Receive1_Tests (Socket, Expected => Bad_File_Descriptor,
      Er1 => "A059", Er2 => "A060",
```

```
      Er3 => "A061", Er4 => "A062");
end;

------------------------------------------------------------------------
--  Trying to receive on a non-connected socket generates
--  the Not_Connected error code.

Test ("Receive 1: Not_Connected [18.4.12.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Receive1_Tests (Socket, Expected => Not_Connected,
      Er1 => "A063", Er2 => "A064",
      Er3 => "A065", Er4 => "A066");
   Close (Socket);
end;

------------------------------------------------------------------------
--  Trying to receive on a non-socket generates
--  the Not_A_Socket error code.

Test ("Receive 1: Not_A_Socket [18.4.12.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 1;
begin
   Receive1_Tests (Socket, Expected => Not_A_Socket,
      Er1 => "A067", Er2 => "A068",
      Er3 => "A069", Er4 => "A070");
end;

------------------------------------------------------------------------
--  Trying to receive using the Process_OOB_Data option on a socket
--  type that doesn't allow it generate the
--  Option_Not_Supported error code.

Test ("Receive 1: Option_Not_Supported [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Receive1_Tests (Socket, Options => Process_OOB_Data,
      Expected => Option_Not_Supported,
      Er1 => "A080", Er2 => "A081",
      Er3 => "A082", Er4 => "A083");
   Close (Socket);
end;

------------------------------------------------------------------------
--  Receive a message sent to local socket

Test ("Receive 2 [18.4.12]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Port      : Positive;
   Template  : Process_Template;
   Child_ID  : Process_ID;
   Args      : POSIX_String_List;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
   Connection_Socket : POSIX_IO.File_Descriptor := 0;
begin
   Set_Internet_Port (Int_Add, Unspecified_Internet_Port);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Bind (Socket, +Name);
   Port := Integer (Get_Internet_Port (Get_Socket_Name (Socket)));
   Listen (Socket);

   Comment ("port is " & Integer'Image (Port));
   Make_Empty (Args);
   POSIX.Append (Args, ChildE_Name);
   POSIX.Append (Args, "-child");
   POSIX.Append (Args, To_POSIX_String (Integer'Image
      (Port)));
   Pass_Through_Verbosity (Args);
   Open_Template (Template);
   Comment ("Starting ChildE process");
   Start_Process (Child_ID, ChildE_Pathname, Template, Args);
   Close_Template (Template);
   Connection_Socket := Accept_Connection (Socket);
   Receive2_Tests (Connection_Socket, Er1 => "",
      Er2 => "A084", Er3 => "A085", Er4 => "A086");
   Close (Socket);
end;

------------------------------------------------------------------------
--  A non-blocking socket that should block on a Receive generates
--  the Would_Block error code

Test ("Receive 2: Would_Block [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, 9);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Set_File_Control (Socket, Non_Blocking);
   Connect (Socket, +Name);
   Receive2_Tests (Socket, Expected => Would_Block,
      Er1 => "A087", Er2 => "A088",
      Er3 => "A089", Er4 => "A090");
   Close (Socket);
end;

------------------------------------------------------------------------
--  Trying to receive on a closed file descriptor generates
--  the Bad_File_Desriptor error code.

Test ("Receive 2: Bad_File_Destriptor [18.4.12.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Close (Socket);
   Receive2_Tests (Socket, Expected => Bad_File_Descriptor,
      Er1 => "A091", Er2 => "A092",
      Er3 => "A093", Er4 => "A094");
end;

------------------------------------------------------------------------
--  Trying to receive on a non-connected socket generates
--  the Not_Connected error code.

Test ("Receive 2: Not_Connected [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Receive2_Tests (Socket, Expected => Not_Connected,
      Er1 => "A095", Er2 => "A096",
      Er3 => "A097", Er4 => "A098");
   Close (Socket);
end;

------------------------------------------------------------------------
--  Trying to receive on a non-socket generates
--  the Not_A_Socket error code.

Test ("Receive 2: Not_A_Socket [18.4.12.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 1;
begin
   Receive2_Tests (Socket, Expected => Not_A_Socket,
      Er1 => "A099", Er2 => "A100",
      Er3 => "A101", Er4 => "A102");
end;

------------------------------------------------------------------------
--  Trying to receive using the Process_OOB_Data option on a socket
--  type that doesn't allow it generate the
--  Option_Not_Supported error code.

Test ("Receive 2: Option_Not_Supported [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
begin
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Receive2_Tests (Socket, Options => Process_OOB_Data,
      Expected => Option_Not_Supported,
      Er1 => "A103", Er2 => "A104",
      Er3 => "A105", Er4 => "A106");
   Close (Socket);
end;

------------------------------------------------------------------------
--  Receive a message sent to local socket

Test ("Receive 3 [18.4.12]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Port      : Positive;
   Template  : Process_Template;
   Child_ID  : Process_ID;
   Args      : POSIX_String_List;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, Unspecified_Internet_Port);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Bind (Socket, +Name);
   Port := Integer (Get_Internet_Port (Get_Socket_Name (Socket)));

   Comment ("port is " & Integer'Image (Port));
   Make_Empty (Args);
   POSIX.Append (Args, ChildC_Name);
   POSIX.Append (Args, "-child");
   POSIX.Append (Args, To_POSIX_String (Integer'Image
      (Port)));
   Pass_Through_Verbosity (Args);
   Open_Template (Template);
   Comment ("Starting ChildC process");
   Start_Process (Child_ID, ChildC_Pathname, Template, Args);
   Close_Template (Template);
   Receive3_Tests (Socket, From => +Name,
      Er1 => "", Er2 => "A107",
      Er3 => "A108", Er4 => "A109");
   Close (Socket);
end;

------------------------------------------------------------------------
```

```
--  A non-blocking socket that should block on a Receive generates
--  the Would_Block error code

Test ("Receive 3: Would_Block [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, 9);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Set_File_Control (Socket, Non_Blocking);
   Connect (Socket, +Name);
   Receive3_Tests (Socket, From => +Name, Expected => Would_Block,
      Er1 => "A110", Er2 => "A111",
      Er3 => "A112", Er4 => "A113");
   Close (Socket);
end;

------------------------------------------------------------------------
--  Trying to receive on a closed file descriptor generates
--  the Bad_File_Desriptor error code.

Test ("Receive 3: Bad_File_Destriptor [18.4.12.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, 9);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Close (Socket);
   Receive3_Tests (Socket, From => +Name, Expected => Bad_File_Descriptor,
      Er1 => "A114", Er2 => "115",
      Er3 => "A116", Er4 => "117");
end;

------------------------------------------------------------------------
--  Trying to receive on a non-connected socket generates
--  the Not_Connected error code.

Test ("Receive 3: Not_Connected [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Receive3_Tests (Socket, From => +Name, Expected => Not_Connected,
      Er1 => "A118", Er2 => "A119",
      Er3 => "A120", Er4 => "A121");
   Close (Socket);
end;

------------------------------------------------------------------------
--  Trying to receive on a non-socket generates
--  the Not_A_Socket error code.

Test ("Receive 3: Not_A_Socket [18.4.12.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 1;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, 9);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Receive3_Tests (Socket, From => +Name, Expected => Not_A_Socket,
      Er1 => "A122", Er2 => "A123",
      Er3 => "A124", Er4 => "A125");
end;

------------------------------------------------------------------------
--  Trying to receive using the Process_OOB_Data option on a socket
--  type that doesn't allow it generate the
--  Option_Not_Supported error code.

Test ("Receive 3: Option_Not_Supported [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, 9);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Receive3_Tests (Socket, From => +Name, Options => Process_OOB_Data,
      Expected => Option_Not_Supported,
      Er1 => "A126", Er2 => "A127",
      Er3 => "A128", Er4 => "A129");
   Close (Socket);
end;

--  ... Add test for Incorrect_Address_Type here when testing of
--  ... a non-internet protocol is completed

------------------------------------------------------------------------
--  Receive a message sent to local socket

Test ("Receive 4 [18.4.12]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Port      : Positive;
   Template  : Process_Template;
   Child_ID  : Process_ID;
   Args      : POSIX_String_List;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, Unspecified_Internet_Port);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Bind (Socket, +Name);
   Port := Integer (Get_Internet_Port (Get_Socket_Name (Socket)));

   Comment ("port is " & Integer'Image (Port));
   Make_Empty (Args);
   POSIX.Append (Args, ChildG_Name);
   POSIX.Append (Args, "-child");
   POSIX.Append (Args, To_POSIX_String (Integer'Image
      (Port)));
   Pass_Through_Verbosity (Args);
   Open_Template (Template);
   Comment ("Starting ChildG process");
   Start_Process (Child_ID, ChildG_Pathname, Template, Args);
   Close_Template (Template);
   Receive4_Tests (Socket, From => +Name,
      Er1 => "", Er2 => "A130",
      Er3 => "A131", Er4 => "A132");
   Close (Socket);
end;

------------------------------------------------------------------------
--  A non-blocking socket that should block on a Receive generates
--  the Would_Block error code

Test ("Receive 4: Would_Block [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Port (Int_Add, 9);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Set_File_Control (Socket, Non_Blocking);
   Connect (Socket, +Name);
   Receive4_Tests (Socket, From => +Name, Expected => Would_Block,
      Er1 => "A133", Er2 => "A134",
      Er3 => "A135", Er4 => "A136");
   Close (Socket);
end;

------------------------------------------------------------------------
--  Trying to receive on a closed file descriptor generates
--  the Bad_File_Desriptor error code.

Test ("Receive 4: Bad_File_Destriptor [18.4.12.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Close (Socket);
   Receive4_Tests (Socket, From => +Name, Expected => Bad_File_Descriptor,
      Er1 => "A137", Er2 => "A138",
      Er3 => "A139", Er4 => "A140");
end;

------------------------------------------------------------------------
--  Trying to receive on a non-connected socket generates
--  the Not_Connected error code.

Test ("Receive 4: Not_Connected [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Receive4_Tests (Socket, From => +Name, Expected => Not_Connected,
      Er1 => "A141", Er2 => "A142",
      Er3 => "A143", Er4 => "A144");
   Close (Socket);
end;

------------------------------------------------------------------------
```

```
--  Trying to receive on a non-socket generates
--  the Not_A_Socket error code.

Test ("Receive 4: Not_A_Socket [18.4.12.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 1;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Receive4_Tests (Socket, From => +Name, Expected => Not_A_Socket,
      Er1 => "A145", Er2 => "A146",
      Er3 => "A147", Er4 => "A148");
end;

----------------------------------------------------------------------
--  Trying to receive using the Process_OOB_Data option on a socket
--  type that doesn't allow it generates the
--  Option_Not_Supported error code.

Test ("Receive 4: Option_Not_Supported [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
begin
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Receive4_Tests (Socket, From => +Name, Options => Process_OOB_Data,
      Expected => Option_Not_Supported,
      Er1 => "A149", Er2 => "A150",
      Er3 => "A151", Er4 => "A152");
   Close (Socket);
end;

--  ... Add test for Incorrect_Address_Type here when testing of
--  ... a non-internet protocol is completed

----------------------------------------------------------------------
--  Receive a message sent to local socket

Test ("Receive_Message 1 [18.4.12]");
declare
   Socket      : POSIX_IO.File_Descriptor;
   Port        : Positive;
   Template    : Process_Template;
   Child_ID    : Process_ID;
   Args        : POSIX_String_List;
   Int_Add     : aliased Internet_Socket_Address;
   Name        : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
   Message     : Socket_Message;
   Buffer      : POSIX_String (1 .. 80);
   IOV_Array   : IO_Vector_Array_Pointer :=
       new IO_Vector_Array (1 .. 1);
   Recv_Length : POSIX.IO_Count := 0;

begin
   Set_Buffer (IOV_Array (1), Buffer (Buffer'First)'Address,
               Buffer'Length);
   Set_IO_Vector_Array (Message, IOV_Array);
   Set_Internet_Port (Int_Add, Unspecified_Internet_Port);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Bind (Socket, +Name);
   Port := Integer (Get_Internet_Port (Get_Socket_Name (Socket)));

   Comment ("port is " & Integer'Image (Port));
   Make_Empty (Args);
   POSIX.Append (Args, ChildD_Name);
   POSIX.Append (Args, "-child");
   POSIX.Append (Args, To_POSIX_String (Integer'Image
      (Port)));
   Pass_Through_Verbosity (Args);
   Open_Template (Template);
   Comment ("Starting ChildD process");
   Start_Process (Child_ID, ChildD_Pathname, Template, Args);
   Close_Template (Template);
   Receive_Message1_Tests (Socket, Message,
      Er1 => "", Er2 => "A153",
      Er3 => "A154", Er4 => "A155", Received_Length => Recv_Length);
   Comment (To_String (Buffer (1 .. Integer (Recv_Length))));
   Close (Socket);
end;

----------------------------------------------------------------------
--  A non-blocking socket that should block on a Receive generates
--  the Would_Block error code

Test ("Receive_Message 1: Would_Block [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Message   : Socket_Message;
   Buffer    : POSIX_String (1 .. 80);
   IOV_Array : IO_Vector_Array_Pointer :=
       new IO_Vector_Array (1 .. 1);
   Recv_Length : POSIX.IO_Count := 0;
begin
   Set_Buffer (IOV_Array (1), Buffer (Buffer'First)'Address,
               Buffer'Length);
   Set_IO_Vector_Array (Message, IOV_Array);
   Set_Internet_Port (Int_Add, 9);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Set_File_Control (Socket, Non_Blocking);
   Receive_Message1_Tests (Socket, Message, Expected => Would_Block,
      Er1 => "A156", Er2 => "A157",
      Er3 => "A158", Er4 => "A159", Received_Length => Recv_Length);
   Close (Socket);
end;

----------------------------------------------------------------------
--  Trying to receive on a closed file descriptor generates
--  the Bad_File_Desriptor error code.

Test ("Receive_Message 1: Bad_File_Destriptor [18.4.12.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Message   : Socket_Message;
   Buffer    : POSIX_String (1 .. 80);
   IOV_Array : IO_Vector_Array_Pointer :=
       new IO_Vector_Array (1 .. 1);
   Recv_Length : POSIX.IO_Count := 0;
begin
   Set_Buffer (IOV_Array (1), Buffer (Buffer'First)'Address,
               Buffer'Length);
   Set_IO_Vector_Array (Message, IOV_Array);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Close (Socket);
   Receive_Message1_Tests (Socket, Message, Expected => Bad_File_Descriptor,
      Er1 => "A160", Er2 => "A161",
      Er3 => "A162", Er4 => "A163", Received_Length => Recv_Length);
end;

----------------------------------------------------------------------
--  Trying to receive on a non-connected socket generates
--  the Not_Connected error code.

Test ("Receive_Message 1: Not_Connected [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Message   : Socket_Message;
   Buffer    : POSIX_String (1 .. 80);
   IOV_Array : IO_Vector_Array_Pointer :=
       new IO_Vector_Array (1 .. 1);
   Recv_Length : POSIX.IO_Count := 0;
begin
   Set_Buffer (IOV_Array (1), Buffer (Buffer'First)'Address,
               Buffer'Length);
   Set_IO_Vector_Array (Message, IOV_Array);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Receive_Message1_Tests (Socket, Message, Expected => Not_Connected,
      Er1 => "A164", Er2 => "A165",
      Er3 => "A166", Er4 => "A167", Received_Length => Recv_Length);
   Close (Socket);
end;

----------------------------------------------------------------------
--  Trying to receive on a non-socket generates
--  the Not_A_Socket error code.

Test ("Receive_Message 1: Not_A_Socket [18.4.12.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 1;
   Int_Add   : aliased Internet_Socket_Address;
   Message   : Socket_Message;
   Buffer    : POSIX_String (1 .. 80);
   IOV_Array : IO_Vector_Array_Pointer :=
       new IO_Vector_Array (1 .. 1);
   Recv_Length : POSIX.IO_Count := 0;
begin
   Set_Buffer (IOV_Array (1), Buffer (Buffer'First)'Address,
               Buffer'Length);
   Set_IO_Vector_Array (Message, IOV_Array);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Receive_Message1_Tests (Socket, Message, Expected => Not_A_Socket,
      Er1 => "A168", Er2 => "A169",
      Er3 => "A170", Er4 => "A171", Received_Length => Recv_Length);
end;

----------------------------------------------------------------------
--  Trying to receive using the Process_OOB_Data option on a socket
--  type that doesn't allow it generates the
--  Option_Not_Supported error code.

Test ("Receive_Message 1: Option_Not_Supported [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Message   : Socket_Message;
   Buffer    : POSIX_String (1 .. 80);
   IOV_Array : IO_Vector_Array_Pointer :=
       new IO_Vector_Array (1 .. 1);
   Recv_Length : POSIX.IO_Count := 0;
begin
   Set_Buffer (IOV_Array (1), Buffer (Buffer'First)'Address,
```

```
               Buffer'Length);
   Set_IO_Vector_Array (Message, IOV_Array);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Receive_Message1_Tests (Socket, Message, Options => Process_OOB_Data,
      Expected => Option_Not_Supported,
      Er1 => "A172", Er2 => "A173",
      Er3 => "A174", Er4 => "A175", Received_Length => Recv_Length);
   Close (Socket);
end;

--  ... Add test for Incorrect_Address_Type here when testing of
--  ... a non-internet protocol is completed

------------------------------------------------------------------------
--  Receive a message sent to local socket

Test ("Receive_Message 2 [18.4.12]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Port      : Positive;
   Template  : Process_Template;
   Child_ID  : Process_ID;
   Args      : POSIX_String_List;
   Int_Add   : aliased Internet_Socket_Address;
   Name      : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
   Message   : Socket_Message;
   Buffer    : POSIX_String (1 .. 80);
   IOV_Array : IO_Vector_Array_Pointer :=
       new IO_Vector_Array (1 .. 1);
   Recv_Length : POSIX.IO_Count := 0;

begin
   Set_Buffer (IOV_Array (1), Buffer (Buffer'First)'Address,
               Buffer'Length);
   Set_IO_Vector_Array (Message, IOV_Array);
   Set_Internet_Port (Int_Add, Unspecified_Internet_Port);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Bind (Socket, +Name);
   Port := Integer (Get_Internet_Port (Get_Socket_Name (Socket)));

   Comment ("port is " & Integer'Image (Port));
   Make_Empty (Args);
   POSIX.Append (Args, ChildF_Name);
   POSIX.Append (Args, "-child");
   POSIX.Append (Args, To_POSIX_String (Integer'Image
      (Port)));
   Pass_Through_Verbosity (Args);
   Open_Template (Template);
   Comment ("Starting ChildF process");
   Start_Process (Child_ID, ChildF_Pathname, Template, Args);
   Close_Template (Template);
   Receive_Message2_Tests (Socket, Message,
      Er1 => "", Er2 => "A176",
      Er3 => "A177", Er4 => "A178", Received_Length => Recv_Length);
   Comment (To_String (Buffer (1 .. Integer (Recv_Length))));
   Close (Socket);
end;

------------------------------------------------------------------------
--  A non-blocking socket that should block on a Receive generates
--  the Would_Block error code

Test ("Receive_Message 2: Would_Block [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Message   : Socket_Message;
   Buffer    : POSIX_String (1 .. 80);
   IOV_Array : IO_Vector_Array_Pointer :=
       new IO_Vector_Array (1 .. 1);
   Recv_Length : POSIX.IO_Count := 0;
begin
   Set_Buffer (IOV_Array (1), Buffer (Buffer'First)'Address,
               Buffer'Length);
   Set_IO_Vector_Array (Message, IOV_Array);
   Set_Internet_Port (Int_Add, 9);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Set_File_Control (Socket, Non_Blocking);
   Receive_Message2_Tests (Socket, Message, Expected => Would_Block,
      Er1 => "A179", Er2 => "A180",
      Er3 => "A181", Er4 => "A182", Received_Length => Recv_Length);
   Close (Socket);
end;

------------------------------------------------------------------------
--  Trying to receive on a closed file descriptor generates
--  the Bad_File_Desriptor error code.

Test ("Receive_Message 2: Bad_File_Destriptor [18.4.12.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Message   : Socket_Message;
   Buffer    : POSIX_String (1 .. 80);
   IOV_Array : IO_Vector_Array_Pointer :=
       new IO_Vector_Array (1 .. 1);
   Recv_Length : POSIX.IO_Count := 0;
begin
   Set_Buffer (IOV_Array (1), Buffer (Buffer'First)'Address,
               Buffer'Length);
   Set_IO_Vector_Array (Message, IOV_Array);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Close (Socket);
   Receive_Message2_Tests (Socket, Message, Expected => Bad_File_Descriptor,
      Er1 => "A183", Er2 => "A184",
      Er3 => "A185", Er4 => "A186", Received_Length => Recv_Length);
end;

------------------------------------------------------------------------
--  Trying to receive on a non-connected socket generates
--  the Not_Connected error code.

Test ("Receive_Message 2: Not_Connected [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Message   : Socket_Message;
   Buffer    : POSIX_String (1 .. 80);
   IOV_Array : IO_Vector_Array_Pointer :=
       new IO_Vector_Array (1 .. 1);
   Recv_Length : POSIX.IO_Count := 0;
begin
   Set_Buffer (IOV_Array (1), Buffer (Buffer'First)'Address,
               Buffer'Length);
   Set_IO_Vector_Array (Message, IOV_Array);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Receive_Message2_Tests (Socket, Message, Expected => Not_Connected,
      Er1 => "A187", Er2 => "A188",
      Er3 => "A189", Er4 => "A190", Received_Length => Recv_Length);
   Close (Socket);
end;

------------------------------------------------------------------------
--  Trying to receive on a non-socket generates
--  the Not_A_Socket error code.

Test ("Receive_Message 2: Not_A_Socket [18.4.12.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 1;
   Int_Add   : aliased Internet_Socket_Address;
   Message   : Socket_Message;
   Buffer    : POSIX_String (1 .. 80);
   IOV_Array : IO_Vector_Array_Pointer :=
       new IO_Vector_Array (1 .. 1);
   Recv_Length : POSIX.IO_Count := 0;
begin
   Set_Buffer (IOV_Array (1), Buffer (Buffer'First)'Address,
               Buffer'Length);
   Set_IO_Vector_Array (Message, IOV_Array);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Receive_Message2_Tests (Socket, Message, Expected => Not_A_Socket,
      Er1 => "A191", Er2 => "A192",
      Er3 => "A193", Er4 => "A194", Received_Length => Recv_Length);
end;

------------------------------------------------------------------------
--  Trying to receive using the Process_OOB_Data option on a socket
--  type that doesn't allow it generates the
--  Option_Not_Supported error code.

Test ("Receive_Message 2: Option_Not_Supported [18.4.12.3]");
declare
   Socket    : POSIX_IO.File_Descriptor;
   Int_Add   : aliased Internet_Socket_Address;
   Message   : Socket_Message;
   Buffer    : POSIX_String (1 .. 80);
   IOV_Array : IO_Vector_Array_Pointer :=
       new IO_Vector_Array (1 .. 1);
   Recv_Length : POSIX.IO_Count := 0;
begin
   Set_Buffer (IOV_Array (1), Buffer (Buffer'First)'Address,
               Buffer'Length);
   Set_IO_Vector_Array (Message, IOV_Array);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
   Receive_Message2_Tests (Socket, Message, Options => Process_OOB_Data,
      Expected => Option_Not_Supported,
      Er1 => "A195", Er2 => "A196",
      Er3 => "A197", Er4 => "A198", Received_Length => Recv_Length);
   Close (Socket);
end;

--  ... Add test for Incorrect_Address_Type here when testing of
--  ... a non-internet protocol is completed

   Done;
exception when E : others => Fatal_Exception (E, "A999");
end p180402;
```

p180402a.adb

```
------------------------------------------------------------------------------
--                                                                          --
```

```
--                      POSIX.5c VALIDATION TEST SUITE                      --
--                                                                          --
--                             P 1 8 0 4 0 2 A                              --
--                                                                          --
--                                 B o d y                                  --
--                                                                          --
--                                                                          --
--  Copyright (c) 1995-1999 Florida  State  University  (FSU).  All Rights  --
--  Reserved.                                                               --
--                                                                          --
--  This is free software;  you can redistribute it and/or modify it under  --
--  terms of the  GNU  General  Public  License  as published by the  Free  --
--  Software Foundation;  either version 2, or (at your option)  any  later --
--  version.  This  software  is distributed  in the hope that it  will be  --
--  useful, but WITHOUT ANY WARRANTY; without even the implied warranty of  --
--  MERCHANTABILITY   or  FITNESS FOR A PARTICULAR PURPOSE.   See the  GNU  --
--  General Public License for more details.  You  should have  received a  --
--  copy of the GNU General Public License  distributed  with  GNARL;  see  --
--  file  COPYING.  If not,  write to  the  Free  Software  Foundation, 59  --
--  Temple Place - Suite 330, Boston, MA 02111-1307, USA.                   --
--                                                                          --
--  Under contract  GS-35F-4506G, the U. S. Government obtained  unlimited  --
--  rights in the software and documentation contained herein.   Unlimited  --
--  rights are defined in DFAR 252,227-7013(a)(19).  By making this public  --
--  release,   the  Government  intends  to  confer  upon  all  recipients  --
--  unlimited  rights equal to those held by the Government.  These rights  --
--  include rights to use,  duplicate,  release  or  disclose the released  --
--  data an computer software  in whole or in part,  in any manner and for  --
--  any purpose whatsoever, and to have or permit others to do so.          --
--                                                                          --
--  DISCLAIMER   --   ALL MATERIALS OR INFORMATION HEREIN RELEASED,   MADE  --
--  AVAILABLE OR DISCLOSED ARE AS IS.   THE GOVERNMENT MAKES NO EXPRESS OR  --
--  IMPLIED WARRANTY AS TO ANY MATTER WHATSOEVER, INCLUDING THE CONDITIONS  --
--  OF THE SOFTWARE,  DOCUMENTATION  OR  OTHER INFORMATION RELEASED,  MADE  --
--  AVAILABLE OR DISCLOSED,  OR THE OWNERSHIP,  MERCHANTABILITY, OR FITNESS  --
--  FOR A PARTICULAR PURPOSE OF SAID MATERIAL.                              --
--                                                                          --
------------------------------------------------------------------------------

--  Multiprocess integrated test for package POSIX_Sockets
--  in IEEE Std 1003.5c Section 18.4 with package
--  POSIX_Sockets_Interent.

--  This test covers only features that depend only on
--  the packages (POSIX_Sockets/_Interent) and features from
--  other packages that are required to be supported.

with POSIX,
     POSIX_Sockets,
     POSIX_Sockets_Internet,
     POSIX_Report,
     POSIX_Process_Environment,
     POSIX_IO;

procedure p180402a is

   use POSIX,
       POSIX_Sockets,
       POSIX_Sockets_Internet,
       POSIX_IO,
       POSIX_Process_Environment,
       POSIX_Report;

   Socket  : POSIX_IO.File_Descriptor := 0;
   Int_Add : aliased Internet_Socket_Address;
   Name    : Internet_Socket_Address_Pointer :=
      Int_Add'Unchecked_Access;
   Port    : Internet_Port;

---------------------------------------------------------------------------
--  Begin Tests

begin

   ------------------------------------------------------------------------
   --  Make a connection

   Comment ("p180402a");
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Port := Internet_Port (Integer'Value
      (To_String (Value (Argument_List, 3))));
   Comment ("Child A sending to Port " & Internet_Port'Image (Port));
   Set_Internet_Port (Int_Add, Port);
   Set_Internet_Address (Int_Add,
      String_To_Internet_Address ("127.0.0.1"));
   Connect (Socket, +Name);
   Close (Socket);

exception when E : others => Fatal_Exception (E, "A001: p180402a");
end p180402a;
```

p180402b.adb

```
------------------------------------------------------------------------------
--                                                                          --
--                      POSIX.5c VALIDATION TEST SUITE                      --
--                                                                          --
--                             P 1 8 0 4 0 2 B                              --
--                                                                          --
--                                 B o d y                                  --
--                                                                          --
--                                                                          --
--  Copyright (c) 1995-1999 Florida  State  University  (FSU).  All Rights  --
--  Reserved.                                                               --
--                                                                          --
--  This is free software;  you can redistribute it and/or modify it under  --
--  terms of the  GNU  General  Public  License  as published by the  Free  --
--  Software Foundation;  either version 2, or (at your option)  any  later --
--  version.  This  software  is distributed  in the hope that it  will be  --
--  useful, but WITHOUT ANY WARRANTY; without even the implied warranty of  --
--  MERCHANTABILITY   or  FITNESS FOR A PARTICULAR PURPOSE.   See the  GNU  --
--  General Public License for more details.  You  should have  received a  --
--  copy of the GNU General Public License  distributed  with  GNARL;  see  --
--  file  COPYING.  If not,  write to  the  Free  Software  Foundation, 59  --
--  Temple Place - Suite 330, Boston, MA 02111-1307, USA.                   --
--                                                                          --
--  Under contract  GS-35F-4506G, the U. S. Government obtained  unlimited  --
--  rights in the software and documentation contained herein.   Unlimited  --
--  rights are defined in DFAR 252,227-7013(a)(19).  By making this public  --
--  release,   the  Government  intends  to  confer  upon  all  recipients  --
--  unlimited  rights equal to those held by the Government.  These rights  --
--  include rights to use,  duplicate,  release  or  disclose the released  --
--  data an computer software  in whole or in part,  in any manner and for  --
--  any purpose whatsoever, and to have or permit others to do so.          --
--                                                                          --
--  DISCLAIMER   --   ALL MATERIALS OR INFORMATION HEREIN RELEASED,   MADE  --
--  AVAILABLE OR DISCLOSED ARE AS IS.   THE GOVERNMENT MAKES NO EXPRESS OR  --
--  IMPLIED WARRANTY AS TO ANY MATTER WHATSOEVER, INCLUDING THE CONDITIONS  --
--  OF THE SOFTWARE,  DOCUMENTATION  OR  OTHER INFORMATION RELEASED,  MADE  --
--  AVAILABLE OR DISCLOSED,  OR THE OWNERSHIP,  MERCHANTABILITY, OR FITNESS  --
--  FOR A PARTICULAR PURPOSE OF SAID MATERIAL.                              --
--                                                                          --
------------------------------------------------------------------------------

--  Multiprocess integrated test for package POSIX_Sockets
--  in IEEE Std 1003.5c Section 18.4 with package
--  POSIX_Sockets_Interent.

--  This test covers only features that depend only on
--  the packages (POSIX_Sockets/_Interent) and features from
--  other packages that are required to be supported.

with POSIX,
     POSIX_Sockets,
     POSIX_Sockets_Internet,
     POSIX_Report,
     POSIX_Process_Environment,
     System,
     POSIX_IO;


procedure p180402b is

   use POSIX,
       POSIX_Sockets,
       POSIX_Sockets_Internet,
       POSIX_IO,
       POSIX_Process_Environment,
       POSIX_Report;

   --  P180402 expects the following message
   Expected_Message : POSIX_String (1 .. 11) := "Send 1 test";

   procedure Send1_Tests (
      Socket   : in POSIX_IO.File_Descriptor;
      Buffer   : in System.Address;
      Options  : in Message_Option_Set := Empty_Set;
      Mask     : POSIX.Signal_Masking := All_Signals;
      Expected : in Error_Code := No_Error;
      Er1, Er2, Er3 : in String := "");

   procedure Send1_Tests (
      Socket   : in POSIX_IO.File_Descriptor;
      Buffer   : in System.Address;
      Options  : in Message_Option_Set := Empty_Set;
      Mask     : POSIX.Signal_Masking := All_Signals;
      Expected : in Error_Code := No_Error;
      Er1, Er2, Er3 : in String := "")
   is
      Sent    : POSIX.IO_Count;
   begin
      Send (Socket, Buffer, 11, Sent, Mask);
      if Er1 /= "" then
         Expect_Exception (Er1);
      else
         Comment ("Sent " & Integer'Image (Integer (Sent)) & " octets");
      end if;
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Expected then
         Unexpected_Exception (E1, Er2);
      end if;
   when E2 : others => Unexpected_Exception (E2, Er3);
   end Send1_Tests;

---------------------------------------------------------------------------
--  Begin Tests

begin

   ------------------------------------------------------------------------
   --  Send on a socket

   Test ("Send 1 [18.4.13]");
   declare
      Socket  : POSIX_IO.File_Descriptor := 0;
```

```
   Int_Add : aliased Internet_Socket_Address;
   Name    : Internet_Socket_Address_Pointer :=
     Int_Add'Unchecked_Access;
   Port    : Internet_Port;
   Buffer  : POSIX_String (1 .. 11) := Expected_Message;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Port := Internet_Port (Integer'Value
     (To_String (Value (Argument_List, 3))));
   Comment ("Child B sending to port " & Internet_Port'Image (Port));
   Set_Internet_Port (Int_Add, Port);
   Set_Internet_Address (Int_Add,
     String_To_Internet_Address ("127.0.0.1"));
   Connect (Socket, +Name);
   Send1_Tests (Socket, Buffer'Address);
   Close (Socket);
exception
   when E : others => Unexpected_Exception (E, "A001: p180402b");
end;

------------------------------------------------------------------------
--  Trying to send on a closed file descriptor generates
--  the Bad_File_Desriptor error code.

Test ("Send 1: Bad_File_Descriptor [18.4.13.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 0;
   Buffer : POSIX_String := "test";
begin
   Close (Socket);
   Send1_Tests (Socket, Buffer'Address,
                Expected => Bad_File_Descriptor,
                Er1 => "A002: p180402b",
                Er2 => "A003: p180402b",
                Er3 => "A004: p180402b");
exception
   when E : others => Unexpected_Exception (E, "A005: p180402b");
end;

------------------------------------------------------------------------
--  Trying to send on a socket that is not connected generates
--  the Not_Connected error code.

Test ("Send 1: Not_Connected [18.4.13.3]");
declare
   Socket : POSIX_IO.File_Descriptor;
   Buffer : POSIX_String := "test";
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Send1_Tests (Socket, Buffer'Address,
                Expected => Not_Connected,
                Er1 => "A006: p180402b",
                Er2 => "A007: p180402b",
                Er3 => "A008: p180402b");
   Close (Socket);
exception
   when E : others => Unexpected_Exception (E, "A009: p180402b");
end;

------------------------------------------------------------------------
--  Trying to send on a file descriptor that is not a socket generates
--  the Not_A_Socket error code.

Test ("Send 1: Not_A_Socket [18.4.13.3]");
declare
   Socket : POSIX_IO.File_Descriptor := 1;
   Buffer : POSIX_String := "test";
begin
   Send1_Tests (Socket, Buffer'Address,
                Expected => Not_A_Socket,
                Er1 => "A010: p180402b",
                Er2 => "A011: p180402b",
                Er3 => "A012: p180402b");
   Close (Socket);
exception
   when E : others => Unexpected_Exception (E, "A013: p180402b");
end;



------------------------------------------------------------------------
--  Trying to send on a file descriptor that is not a socket generates
--  the Not_A_Socket error code.

Test ("Send 1: Is_Already_Connected [18.4.13.3]");
declare
   Socket  : POSIX_IO.File_Descriptor := 0;
   Int_Add : aliased Internet_Socket_Address;
   Name    : Internet_Socket_Address_Pointer :=
     Int_Add'Unchecked_Access;
   Port    : Internet_Port;
begin
   Socket := Create (Internet_Protocol, Stream_Socket, TCP);
   Port := Internet_Port (Integer'Value
     (To_String (Value (Argument_List, 3))));
   Set_Internet_Port (Int_Add, Port);
   Set_Internet_Address (Int_Add,
     String_To_Internet_Address ("127.0.0.1"));
   Connect (Socket, +Name);
   Connect (Socket, +Name);
   Close (Socket);
exception
when E1 : POSIX_Error =>
   if Get_Error_Code /= Is_Already_Connected then
      Unexpected_Exception (E1, "A014: p180402b");
      end if;
   when E2 : others => Unexpected_Exception (E2, "A015: p180402b");
   end;

exception when E : others => Fatal_Exception (E, "A016: p180402b");
end p180402b;
```

p180402c.adb

```
------------------------------------------------------------------------------
--                                                                          --
--                      POSIX.5c VALIDATION TEST SUITE                      --
--                                                                          --
--                             P 1 8 0 4 0 2 C                              --
--                                                                          --
--                                 B o d y                                  --
--                                                                          --
--                                                                          --
--  Copyright (c) 1995-1999 Florida  State  University  (FSU).  All Rights  --
--  Reserved.                                                               --
--                                                                          --
--  This is free software;  you can redistribute it and/or modify it under  --
--  terms of the  GNU  General  Public  License  as published by the Free   --
--  Software Foundation;  either version 2, or (at your option) any  later  --
--  version.  This  software  is distributed  in the hope that it  will be  --
--  useful, but WITHOUT ANY WARRANTY; without even the implied warranty of  --
--  MERCHANTABILITY   or  FITNESS FOR A PARTICULAR PURPOSE.   See the  GNU  --
--  General Public License for more details.  You  should have  received a  --
--  copy of the GNU General Public License  distributed  with  GNARL;  see  --
--  file  COPYING.  If not,  write to  the  Free  Software  Foundation, 59  --
--  Temple Place - Suite 330, Boston, MA 02111-1307, USA.                   --
--                                                                          --
--  Under contract  GS-35F-4506G, the U. S. Government obtained  unlimited  --
--  rights in the software and documentation contained herein.   Unlimited  --
--  rights are defined in DFAR 252,227-7013(a)(19).  By making this public  --
--  release,   the  Government  intends  to  confer  upon  all  recipients  --
--  unlimited  rights equal to those held by the Government.  These rights  --
--  include rights to use,  duplicate,  release  or  disclose the released  --
--  data an computer software  in whole or in part,  in any manner and for  --
--  any purpose whatsoever, and to have or permit others to do so.          --
--                                                                          --
--  DISCLAIMER   --   ALL MATERIALS OR INFORMATION HEREIN RELEASED,   MADE  --
--  AVAILABLE OR DISCLOSED ARE AS IS.   THE GOVERNMENT MAKES NO EXPRESS OR  --
--  IMPLIED WARRANTY AS TO ANY MATTER WHATSOEVER, INCLUDING THE CONDITIONS  --
--  OF THE SOFTWARE,  DOCUMENTATION  OR  OTHER INFORMATION RELEASED,  MADE  --
--  AVAILABLE OR DISCLOSED,  OR THE OWNERSHIP,  MERCHANTABILITY, OR FITNESS  --
--  FOR A PARTICULAR PURPOSE OF SAID MATERIAL.                              --
--                                                                          --
------------------------------------------------------------------------------

--  Multiprocess integrated test for package POSIX_Sockets
--  in IEEE Std 1003.5c Section 18.4 with package
--  POSIX_Sockets_Interent.

--  This test covers only features that depend only on
--  the packages (POSIX_Sockets/_Interent) and features from
--  other packages that are required to be supported.

with POSIX,
     POSIX_Sockets,
     POSIX_Sockets_Internet,
     POSIX_Report,
     POSIX_Process_Environment,
     System,
     POSIX_IO;

procedure p180402c is

   use POSIX,
       POSIX_Sockets,
       POSIX_Sockets_Internet,
       POSIX_IO,
       POSIX_Process_Environment,
       POSIX_Report;

   --  P180402 expects the following message
   Expected_Message : POSIX_String (1 .. 11) := "Send 3 test";

   procedure Send3_Tests (
      Socket   : in POSIX_IO.File_Descriptor;
      Buffer   : in System.Address;
      Options  : in Message_Option_Set := Empty_Set;
      To       : in Socket_Address_Pointer
               := Null_Socket_Address;
      Mask     : POSIX.Signal_Masking := All_Signals;
      Expected : in Error_Code := No_Error;
      Er1, Er2, Er3 : in String := "");

   procedure Send3_Tests (
      Socket   : in POSIX_IO.File_Descriptor;
      Buffer   : in System.Address;
      Options  : in Message_Option_Set := Empty_Set;
      To       : in Socket_Address_Pointer
               := Null_Socket_Address;
      Mask     : POSIX.Signal_Masking := All_Signals;
      Expected : in Error_Code := No_Error;
      Er1, Er2, Er3 : in String := "")
   is
      Sent    : POSIX.IO_Count;
   begin
      Send (Socket, Buffer, 11, Sent, To, Mask);
```

```
      if Er1 /= "" then
         Expect_Exception (Er1);
      else
         Comment ("Sent " & Integer'Image (Integer (Sent)) & " octets");
      end if;
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Expected then
         Unexpected_Exception (E1, Er2);
      end if;
   when E2 : others => Unexpected_Exception (E2, Er3);
   end Send3_Tests;

----------------------------------------------------------------------------
--  Begin Tests

begin
   -------------------------------------------------------------------------
   --  Send on a socket

   Test ("Send 3 [18.4.13]");
   declare
      Socket  : POSIX_IO.File_Descriptor := 0;
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
      Port    : Internet_Port;
      Buffer  : POSIX_String (1 .. 11) := Expected_Message;
   begin
      Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
      Port := Internet_Port (Integer'Value
        (To_String (Value (Argument_List, 3))));
      Set_Internet_Port (Int_Add, Port);
      Set_Internet_Address (Int_Add,
        String_To_Internet_Address ("127.0.0.1"));
      Comment ("Child C sending to port " & Internet_Port'Image (Port));
      Send3_Tests (Socket, Buffer'Address, To => +Name);
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A001: p180402c");
   end;

   -------------------------------------------------------------------------
   --  Trying to send on a closed file descriptor generates
   --  the Bad_File_Desriptor error code.

   Test ("Send 3: Bad_File_Descriptor [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor := 0;
      Buffer : POSIX_String := "test";
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
   begin
      Set_Internet_Address (Int_Add,
        String_To_Internet_Address ("127.0.0.1"));
      Close (Socket);
      Send3_Tests (Socket, Buffer'Address, To => +Name,
                   Expected => Bad_File_Descriptor,
                   Er1 => "A002: p180402c",
                   Er2 => "A004: p180402c",
                   Er3 => "A005: p180402c");
   exception
      when E : others => Unexpected_Exception (E, "A006: p180402c");
   end;

   -------------------------------------------------------------------------
   --  Trying to send on a socket that is not connected generates
   --  the Not_Connected error code.

   Test ("Send 3: Not_Connected [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor;
      Buffer : POSIX_String := "test";
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
   begin
      Set_Internet_Address (Int_Add,
         String_To_Internet_Address ("127.0.0.1"));
      Socket := Create (Internet_Protocol, Stream_Socket, TCP);
      Send3_Tests (Socket, Buffer'Address, To => +Name,
                   Expected => Not_Connected,
                   Er1 => "A007: p180402c",
                   Er2 => "A008: p180402c",
                   Er3 => "A009: p180402c");
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A010: p180402c");
   end;

   -------------------------------------------------------------------------
   --  Trying to send on a file descriptor that is not a socket generates
   --  the Not_A_Socket error code.

   Test ("Send 3: Not_A_Socket [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor := 1;
      Buffer : POSIX_String := "test";
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
   begin
      Set_Internet_Address (Int_Add,
         String_To_Internet_Address ("127.0.0.1"));
      Send3_Tests (Socket, Buffer'Address, To => +Name,
                   Expected => Not_A_Socket,
                   Er1 => "A011: p180402c",
                   Er2 => "A012: p180402c",
                   Er3 => "A013: p180402c");
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A014: p180402c");
   end;

exception when E : others => Fatal_Exception (E, "A021: p180402c");
end p180402c;
```

p180402d.adb

```
------------------------------------------------------------------------------
--                                                                          --
--                      POSIX.5c VALIDATION TEST SUITE                      --
--                                                                          --
--                             P 1 8 0 4 0 2 D                              --
--                                                                          --
--                                 B o d y                                  --
--                                                                          --
--                                                                          --
--  Copyright (c) 1995-1999 Florida  State  University  (FSU).  All Rights --
--  Reserved.                                                               --
--                                                                          --
--  This is free software;  you can redistribute it and/or modify it under --
--  terms of the  GNU  General  Public  License  as published by the  Free --
--  Software Foundation;  either version 2, or (at your option) any  later --
--  version.  This  software  is distributed  in the hope that it  will be --
--  useful, but WITHOUT ANY WARRANTY; without even the implied warranty of --
--  MERCHANTABILITY   or  FITNESS FOR A PARTICULAR PURPOSE.   See the  GNU --
--  General Public License for more details.  You  should have  received a --
--  copy of the GNU General Public License  distributed  with  GNARL;  see --
--  file  COPYING.  If not,  write to  the  Free  Software  Foundation, 59 --
--  Temple Place - Suite 330, Boston, MA 02111-1307, USA.                  --
--                                                                          --
--  Under contract  GS-35F-4506G, the U. S. Government obtained  unlimited --
--  rights in the software and documentation contained herein.   Unlimited --
--  rights are defined in DFAR 252,227-7013(a)(19).  By making this public --
--  release,   the  Government  intends  to  confer  upon  all  recipients --
--  unlimited  rights equal to those held by the Government.  These rights --
--  include rights to use,  duplicate,  release  or  disclose the released --
--  data an computer software  in whole or in part,  in any manner and for --
--  any purpose whatsoever, and to have or permit others to do so.         --
--                                                                          --
--  DISCLAIMER   --   ALL MATERIALS OR INFORMATION HEREIN RELEASED,   MADE --
--  AVAILABLE OR DISCLOSED ARE AS IS.   THE GOVERNMENT MAKES NO EXPRESS OR --
--  IMPLIED WARRANTY AS TO ANY MATTER WHATSOEVER, INCLUDING THE CONDITIONS --
--  OF THE SOFTWARE,  DOCUMENTATION  OR  OTHER INFORMATION RELEASED,  MADE --
--  AVAILABLE OR DISCLOSED,  OR THE OWNERSHIP,  MERCHANTABILITY, OR FITNESS --
--  FOR A PARTICULAR PURPOSE OF SAID MATERIAL.                              --
--                                                                          --
------------------------------------------------------------------------------

--  Multiprocess integrated test for package POSIX_Sockets
--  in IEEE Std 1003.5c Section 18.4 with package
--  POSIX_Sockets_Interent.

--  This test covers only features that depend only on
--  the packages (POSIX_Sockets/_Interent) and features from
--  other packages that are required to be supported.

with POSIX,
     POSIX_Sockets,
     POSIX_Sockets_Internet,
     POSIX_Report,
     POSIX_Process_Environment,
     POSIX_IO;

procedure p180402d is

   use POSIX,
       POSIX_Sockets,
       POSIX_Sockets_Internet,
       POSIX_IO,
       POSIX_Process_Environment,
       POSIX_Report;

   procedure Send_Message1_Tests (
      Socket   : in POSIX_IO.File_Descriptor;
      Message  : Socket_Message;
      Options  : in Message_Option_Set := Empty_Set;
      Mask     : POSIX.Signal_Masking := All_Signals;
      Expected : in Error_Code := No_Error;
      Er1, Er2, Er3 : in String := "");

   procedure Send_Message1_Tests (
      Socket   : in POSIX_IO.File_Descriptor;
      Message  : Socket_Message;
      Options  : in Message_Option_Set := Empty_Set;
      Mask     : POSIX.Signal_Masking := All_Signals;
      Expected : in Error_Code := No_Error;
      Er1, Er2, Er3 : in String := "")
   is
      Sent    : POSIX.IO_Count;
   begin
      Send_Message (Socket, Message, Sent, Mask);
      if Er1 /= "" then
         Expect_Exception (Er1);
```

```
      else
         Comment ("Sent " & Integer'Image (Integer (Sent)) & " octets");
      end if;
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Expected then
         Unexpected_Exception (E1, Er2);
      end if;
   when E2 : others => Unexpected_Exception (E2, Er3);
   end Send_Message1_Tests;

---------------------------------------------------------------------------
--  Begin Tests

begin

   -----------------------------------------------------------------------
   --  Send a message on a connection-less socket

   Test ("Send Message 1 [18.4.13]");
   declare
      Socket  : POSIX_IO.File_Descriptor := 0;
      Message  : Socket_Message;
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
      Port    : Internet_Port;

      Buffer1        : POSIX_String := "Test message. ";
      Buffer2        : POSIX_String := "This is a UDP/IP message. ";
      Buffer3        : POSIX_String := "It was sent using Send_Message1. ";
      IOV_Array      : IO_Vector_Array_Pointer :=
      new IO_Vector_Array (1 .. 3);

   begin
      Set_Buffer (IOV_Array (1), Buffer1 (Buffer1'First)'Address,
                  Buffer1'Length);
      Set_Buffer (IOV_Array (2), Buffer2 (Buffer2'First)'Address,
                  Buffer2'Length);
      Set_Buffer (IOV_Array (3), Buffer3 (Buffer3'First)'Address,
                  Buffer3'Length);

      Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
      Port := Internet_Port (Integer'Value
               (To_String (Value (Argument_List, 3))));
      Set_Internet_Port (Int_Add, Port);
      Set_Internet_Address (Int_Add,
               String_To_Internet_Address ("127.0.0.1"));
      Set_Socket_Name (Message, +Name);
      Set_IO_Vector_Array (Message, IOV_Array);
      Comment ("Child D sending to port " & Internet_Port'Image (Port));
      Send_Message1_Tests (Socket, Message, Er2 => "A001: p180402d",
                           Er3 => "A002: p180402d");
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A003: p180402d");
   end;

   -----------------------------------------------------------------------
   --  Trying to send on a closed file descriptor generates
   --  the Bad_File_Desriptor error code.

   Test ("Send Message 1: Bad_File_Descriptor [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor := 0;
      Message  : Socket_Message;
   begin
      Close (Socket);
      Send_Message1_Tests (Socket, Message,
                  Expected => Bad_File_Descriptor,
                  Er1 => "A004: p180402d",
                  Er2 => "A005: p180402d",
                  Er3 => "A006: p180402d");
   exception
      when E : others => Unexpected_Exception (E, "A007: p180402d");
   end;

   -----------------------------------------------------------------------
   --  Trying to send on a socket that is not connected generates
   --  the Not_Connected error code.

   Test ("Send Message 1: Not_Connected [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor;
      Message  : Socket_Message;
   begin
      Socket := Create (Internet_Protocol, Stream_Socket, TCP);
      Send_Message1_Tests (Socket, Message,
                  Expected => Not_Connected,
                  Er1 => "A008: p180402d",
                  Er2 => "A009: p180402d",
                  Er3 => "A010: p180402d");
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A011: p180402d");
   end;


   -----------------------------------------------------------------------
   --  Trying to send on a file descriptor that is not a socket generates
   --  the Not_A_Socket error code.

   Test ("Send Message 1: Not_A_Socket [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor := 1;
      Message  : Socket_Message;
   begin
      Send_Message1_Tests (Socket, Message,
                  Expected => Not_A_Socket,
                  Er1 => "A012: p180402d",
                  Er2 => "A013: p180402d",
                  Er3 => "A014: p180402d");
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A015: p180402d");
   end;

exception when E : others => Fatal_Exception (E, "A016: p180402d");
end p180402d;
```

p180402e.adb

```
------------------------------------------------------------------------------
--                                                                          --
--                       POSIX.5c VALIDATION TEST SUITE                     --
--                                                                          --
--                             P 1 8 0 4 0 2 E                              --
--                                                                          --
--                                 B o d y                                  --
--                                                                          --
--                                                                          --
--  Copyright (c) 1995-1999 Florida  State  University  (FSU).  All Rights  --
--  Reserved.                                                               --
--                                                                          --
--  This is free software;  you can redistribute it and/or modify it under  --
--  terms of the  GNU  General  Public  License  as published by the  Free  --
--  Software Foundation;  either version 2, or (at your option)  any  later  --
--  version.  This  software  is distributed  in the hope that it  will be  --
--  useful, but WITHOUT ANY WARRANTY; without even the implied warranty of  --
--  MERCHANTABILITY   or  FITNESS FOR A PARTICULAR PURPOSE.   See the  GNU  --
--  General Public License for more details.  You  should have  received a  --
--  copy of the GNU General Public License  distributed  with  GNARL;  see  --
--  file  COPYING.  If not,  write to  the  Free  Software  Foundation, 59  --
--  Temple Place - Suite 330, Boston, MA 02111-1307, USA.                   --
--                                                                          --
--  Under contract  GS-35F-4506G, the U. S. Government obtained  unlimited  --
--  rights in the software and documentation contained herein.   Unlimited  --
--  rights are defined in DFAR 252,227-7013(a)(19).  By making this public  --
--  release,   the  Government  intends  to  confer  upon  all  recipients  --
--  unlimited  rights equal to those held by the Government.  These rights  --
--  include rights to use,  duplicate,  release  or  disclose the released  --
--  data an computer software  in whole or in part,  in any manner and for  --
--  any purpose whatsoever, and to have or permit others to do so.          --
--                                                                          --
--  DISCLAIMER   --   ALL MATERIALS OR INFORMATION HEREIN RELEASED,   MADE  --
--  AVAILABLE OR DISCLOSED ARE AS IS.   THE GOVERNMENT MAKES NO EXPRESS OR  --
--  IMPLIED WARRANTY AS TO ANY MATTER WHATSOEVER, INCLUDING THE CONDITIONS  --
--  OF THE SOFTWARE,  DOCUMENTATION  OR  OTHER INFORMATION RELEASED,  MADE  --
--  AVAILABLE OR DISCLOSED,  OR THE OWNERSHIP,  MERCHANTABILITY, OR FITNESS  --
--  FOR A PARTICULAR PURPOSE OF SAID MATERIAL.                              --
--                                                                          --
------------------------------------------------------------------------------

--  Multiprocess integrated test for package POSIX_Sockets
--  in IEEE Std 1003.5c Section 18.4 with package
--  POSIX_Sockets_Interent.

--  This test covers only features that depend only on
--  the packages (POSIX_Sockets/_Interent) and features from
--  other packages that are required to be supported.

with POSIX,
     POSIX_Sockets,
     POSIX_Sockets_Internet,
     POSIX_Report,
     POSIX_Process_Environment,
     POSIX_IO,
     System;


procedure p180402e is

   use POSIX,
       POSIX_Sockets,
       POSIX_Sockets_Internet,
       POSIX_IO,
       POSIX_Process_Environment,
       POSIX_Report;

   --  P180402 expects the following message
   Expected_Message : POSIX_String (1 .. 11) := "Send 2 test";

   procedure Send2_Tests (
      Socket   : in POSIX_IO.File_Descriptor;
      Buffer   : in System.Address;
      Options  : in Message_Option_Set := Empty_Set;
      Expected : in Error_Code := No_Error;
      Er1, Er2, Er3 : in String := "");

   procedure Send2_Tests (
      Socket   : in POSIX_IO.File_Descriptor;
      Buffer   : in System.Address;
      Options  : in Message_Option_Set := Empty_Set;
      Expected : in Error_Code := No_Error;
      Er1, Er2, Er3 : in String := "")
   is
```

```
      Sent    : POSIX.IO_Count;
   begin
      Send (Socket, Buffer, 11, Sent);
      if Er1 /= "" then
         Expect_Exception (Er1);
      else
         Comment ("Sent " & Integer'Image (Integer (Sent)) & " octets");
      end if;
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Expected then
         Unexpected_Exception (E1, Er2);
      end if;
   when E2 : others => Unexpected_Exception (E2, Er3);
   end Send2_Tests;

----------------------------------------------------------------------------
--  Begin Tests

begin

   -------------------------------------------------------------------------
   --  Send on a socket

   Test ("Send 2 [18.4.13]");
   declare
      Socket  : POSIX_IO.File_Descriptor := 0;
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
      Port    : Internet_Port;
      Buffer  : POSIX_String (1 .. 11) := Expected_Message;
   begin
      Socket := Create (Internet_Protocol, Stream_Socket, TCP);
      Port := Internet_Port (Integer'Value
        (To_String (Value (Argument_List, 3))));
      Set_Internet_Port (Int_Add, Port);
      Set_Internet_Address (Int_Add,
        String_To_Internet_Address ("127.0.0.1"));
      Connect (Socket, +Name);
      Comment ("Child E sending to port " & Internet_Port'Image (Port));
      Send2_Tests (Socket, Buffer'Address);
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A001: p180402e");
   end;

   -------------------------------------------------------------------------
   --  Trying to send on a closed file descriptor generates
   --  the Bad_File_Desriptor error code.

   Test ("Send 2: Bad_File_Descriptor [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor := 0;
      Buffer : POSIX_String := "test";
   begin
      Close (Socket);
      Send2_Tests (Socket, Buffer'Address,
                   Expected => Bad_File_Descriptor,
                   Er1 => "A002: p180402e",
                   Er2 => "A003: p180402e",
                   Er3 => "A004: p180402e");
   exception
      when E : others => Unexpected_Exception (E, "A005: p180402e");
   end;

   -------------------------------------------------------------------------
   --  Trying to send on a socket that is not connected generates
   --  the Not_Connected error code.

   Test ("Send 2: Not_Connected [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor;
      Buffer : POSIX_String := "test";
   begin
      Socket := Create (Internet_Protocol, Stream_Socket, TCP);
      Send2_Tests (Socket, Buffer'Address,
                   Expected => Not_Connected,
                   Er1 => "A006: p180402e",
                   Er2 => "A007: p180402e",
                   Er3 => "A008: p180402e");
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A009: p180402e");
   end;

   -------------------------------------------------------------------------
   --  Trying to send on a file descriptor that is not a socket generates
   --  the Not_A_Socket error code.

   Test ("Send 2: Not_A_Socket [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor := 1;
      Buffer : POSIX_String := "test";
   begin
      Send2_Tests (Socket, Buffer'Address,
                   Expected => Not_A_Socket,
                   Er1 => "A010: p180402e",
                   Er2 => "A011: p180402e",
                   Er3 => "A012: p180402e");
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A013: p180402e");
   end;

   -------------------------------------------------------------------------
   --  Trying to connect to a connected socket generates
   --  the Is_Already_Connected error code.

   Test ("Send 2: Is_Already_Connected");
   declare
      Socket  : POSIX_IO.File_Descriptor := 0;
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
      Port    : Internet_Port;
   begin
      Socket := Create (Internet_Protocol, Stream_Socket, TCP);
      Port := Internet_Port (Integer'Value
        (To_String (Value (Argument_List, 3))));
      Set_Internet_Port (Int_Add, Port);
      Set_Internet_Address (Int_Add,
        String_To_Internet_Address ("127.0.0.1"));
      Connect (Socket, +Name);
      Connect (Socket, +Name);
      Close (Socket);
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Is_Already_Connected then
         Unexpected_Exception (E1, "A014: p180402e");
      end if;
   when E2 : others => Unexpected_Exception (E2, "A015: p180402e");
   end;

exception when E : others => Fatal_Exception (E, "A016: p180402e");
end p180402e;
```

p180402f.adb

```
------------------------------------------------------------------------------
--                                                                          --
--                      POSIX.5c VALIDATION TEST SUITE                      --
--                                                                          --
--                             P 1 8 0 4 0 2 G                              --
--                                                                          --
--                                 B o d y                                  --
--                                                                          --
--                                                                          --
--  Copyright (c) 1995-1999 Florida  State  University  (FSU).  All Rights  --
--  Reserved.                                                               --
--                                                                          --
--  This is free software;  you can redistribute it and/or modify it under  --
--  terms of the  GNU  General  Public  License  as published by the  Free  --
--  Software Foundation;  either version 2, or (at your option)  any  later --
--  version.  This  software  is distributed  in the hope that it  will be  --
--  useful, but WITHOUT ANY WARRANTY; without even the implied warranty of  --
--  MERCHANTABILITY   or  FITNESS FOR A PARTICULAR PURPOSE.   See the  GNU  --
--  General Public License for more details.  You  should have  received a  --
--  copy of the GNU General Public License  distributed  with  GNARL;  see  --
--  file  COPYING.  If not,  write to  the  Free  Software  Foundation, 59  --
--  Temple Place - Suite 330, Boston, MA 02111-1307, USA.                   --
--                                                                          --
--  Under contract  GS-35F-4506G, the U. S. Government obtained  unlimited  --
--  rights in the software and documentation contained herein.   Unlimited  --
--  rights are defined in DFAR 252,227-7013(a)(19).  By making this public  --
--  release,   the  Government  intends  to  confer  upon  all  recipients  --
--  unlimited  rights equal to those held by the Government.  These rights  --
--  include rights to use,  duplicate,  release  or  disclose the released  --
--  data an computer software  in whole or in part,  in any manner and for  --
--  any purpose whatsoever, and to have or permit others to do so.          --
--                                                                          --
--  DISCLAIMER   --   ALL MATERIALS OR INFORMATION HEREIN RELEASED,   MADE  --
--  AVAILABLE OR DISCLOSED ARE AS IS.   THE GOVERNMENT MAKES NO EXPRESS OR  --
--  IMPLIED WARRANTY AS TO ANY MATTER WHATSOEVER, INCLUDING THE CONDITIONS  --
--  OF THE SOFTWARE,  DOCUMENTATION  OR  OTHER INFORMATION RELEASED,  MADE  --
--  AVAILABLE OR DISCLOSED,  OR THE OWNERSHIP,  MERCHANTABILITY, OR FITNESS --
--  FOR A PARTICULAR PURPOSE OF SAID MATERIAL.                              --
--                                                                          --
------------------------------------------------------------------------------

--  Multiprocess integrated test for package POSIX_Sockets
--  in IEEE Std 1003.5c Section 18.4 with package
--  POSIX_Sockets_Interent.

--  This test covers only features that depend only on
--  the packages (POSIX_Sockets/_Interent) and features from
--  other packages that are required to be supported.

with POSIX,
     POSIX_Sockets,
     POSIX_Sockets_Internet,
     POSIX_Report,
     POSIX_Process_Environment,
     System,
     POSIX_IO;

procedure p180402g is

   use POSIX,
       POSIX_Sockets,
       POSIX_Sockets_Internet,
       POSIX_IO,
       POSIX_Process_Environment,
       POSIX_Report;
```

```
   --  P180402 expects the following message
   Expected_Message : POSIX_String (1 .. 11) := "Send 4 test";

   procedure Send4_Tests (
      Socket   : in POSIX_IO.File_Descriptor;
      Buffer   : in System.Address;
      Options  : in Message_Option_Set := Empty_Set;
      To       : in Socket_Address_Pointer
               := Null_Socket_Address;
      Expected : in Error_Code := No_Error;
      Er1, Er2, Er3 : in String := "");

   procedure Send4_Tests (
      Socket   : in POSIX_IO.File_Descriptor;
      Buffer   : in System.Address;
      Options  : in Message_Option_Set := Empty_Set;
      To       : in Socket_Address_Pointer
               := Null_Socket_Address;
      Expected : in Error_Code := No_Error;
      Er1, Er2, Er3 : in String := "")
   is
      Sent    : POSIX.IO_Count;
   begin
      Send (Socket, Buffer, 11, Sent, To);
      if Er1 /= "" then
         Expect_Exception (Er1);
      else
         Comment ("Sent " & Integer'Image (Integer (Sent)) & " octets");
      end if;
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Expected then
         Unexpected_Exception (E1, Er2);
      end if;
   when E2 : others => Unexpected_Exception (E2, Er3);
   end Send4_Tests;

----------------------------------------------------------------------------
--  Begin Tests

begin

   ------------------------------------------------------------------------
   --  Send on a socket

   Test ("Send 4 [18.4.13]");
   declare
      Socket  : POSIX_IO.File_Descriptor := 0;
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
      Port    : Internet_Port;
      Buffer  : POSIX_String (1 .. 11) := Expected_Message;
   begin
      Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
      Port := Internet_Port (Integer'Value
        (To_String (Value (Argument_List, 3))));
      Set_Internet_Port (Int_Add, Port);
      Set_Internet_Address (Int_Add,
        String_To_Internet_Address ("127.0.0.1"));
      Send4_Tests (Socket, Buffer'Address, To => +Name);
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A001: p180402g");
   end;

   ------------------------------------------------------------------------
   --  Trying to send on a closed file descriptor generates
   --  the Bad_File_Desriptor error code.

   Test ("Send 4: Bad_File_Descriptor [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor := 0;
      Buffer : POSIX_String := "test";
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
   begin
      Set_Internet_Address (Int_Add,
        String_To_Internet_Address ("127.0.0.1"));
      Close (Socket);
      Send4_Tests (Socket, Buffer'Address, To => +Name,
                   Expected => Bad_File_Descriptor, Er1 => "A002: p180402g",
                   Er2 => "A003: p180402g", Er3 => "A004: p180402g");
   exception
      when E : others => Unexpected_Exception (E, "A005: p180402g");
   end;

   ------------------------------------------------------------------------
   --  Trying to send on a socket that is not connected generates
   --  the Not_Connected error code.

   Test ("Send 4: Not_Connected [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor;
      Buffer : POSIX_String := "test";
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
   begin
      Set_Internet_Address (Int_Add,
         String_To_Internet_Address ("127.0.0.1"));
      Socket := Create (Internet_Protocol, Stream_Socket, TCP);
      Send4_Tests (Socket, Buffer'Address, To => +Name,
                   Expected => Not_Connected, Er1 => "A006: p180402g",
                   Er2 => "A007: p180402g", Er3 => "A008: p180402g");
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A009: p180402g");
   end;

   ------------------------------------------------------------------------
   --  Trying to send on a file descriptor that is not a socket generates
   --  the Not_A_Socket error code.

   Test ("Send 4: Not_A_Socket [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor := 1;
      Buffer : POSIX_String := "test";
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
   begin
      Set_Internet_Address (Int_Add,
         String_To_Internet_Address ("127.0.0.1"));
      Send4_Tests (Socket, Buffer'Address, To => +Name,
                   Expected => Not_A_Socket, Er1 => "A010: p180402g",
                   Er2 => "A012: p180402g", Er3 => "A013: p180402g");
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A014: p180402g");
   end;

exception when E : others => Fatal_Exception (E, "A017: p180402g");
end p180402g;
```

p180402g.adb

```
------------------------------------------------------------------------------
--                                                                          --
--                      POSIX.5c VALIDATION TEST SUITE                      --
--                                                                          --
--                             P 1 8 0 4 0 2 G                              --
--                                                                          --
--                                 B o d y                                  --
--                                                                          --
--                                                                          --
--  Copyright (c) 1995-1999 Florida  State  University  (FSU).  All Rights  --
--  Reserved.                                                               --
--                                                                          --
--  This is free software;  you can redistribute it and/or modify it under  --
--  terms of the  GNU  General  Public  License  as published by the  Free  --
--  Software Foundation;  either version 2, or (at your option) any  later  --
--  version.  This  software  is distributed  in the hope that it  will be  --
--  useful, but WITHOUT ANY WARRANTY; without even the implied warranty of  --
--  MERCHANTABILITY   or  FITNESS FOR A PARTICULAR PURPOSE.   See the  GNU  --
--  General Public License for more details.  You  should have  received a  --
--  copy of the GNU General Public License  distributed  with  GNARL;  see  --
--  file  COPYING.  If not,  write to  the  Free  Software  Foundation, 59  --
--  Temple Place - Suite 330, Boston, MA 02111-1307, USA.                   --
--                                                                          --
--  Under contract  GS-35F-4506G, the U. S. Government obtained  unlimited  --
--  rights in the software and documentation contained herein.   Unlimited  --
--  rights are defined in DFAR 252,227-7013(a)(19).  By making this public  --
--  release,   the  Government  intends  to  confer  upon  all  recipients  --
--  unlimited  rights equal to those held by the Government.  These rights  --
--  include rights to use,  duplicate,  release  or  disclose the released  --
--  data an computer software  in whole or in part,  in any manner and for  --
--  any purpose whatsoever, and to have or permit others to do so.          --
--                                                                          --
--  DISCLAIMER   --   ALL MATERIALS OR INFORMATION HEREIN RELEASED,   MADE  --
--  AVAILABLE OR DISCLOSED ARE AS IS.   THE GOVERNMENT MAKES NO EXPRESS OR  --
--  IMPLIED WARRANTY AS TO ANY MATTER WHATSOEVER, INCLUDING THE CONDITIONS  --
--  OF THE SOFTWARE,  DOCUMENTATION  OR  OTHER INFORMATION RELEASED,  MADE  --
--  AVAILABLE OR DISCLOSED,  OR THE OWNERSHIP,  MERCHANTABILITY, OR FITNESS --
--  FOR A PARTICULAR PURPOSE OF SAID MATERIAL.                              --
--                                                                          --
------------------------------------------------------------------------------

--  Multiprocess integrated test for package POSIX_Sockets
--  in IEEE Std 1003.5c Section 18.4 with package
--  POSIX_Sockets_Interent.

--  This test covers only features that depend only on
--  the packages (POSIX_Sockets/_Interent) and features from
--  other packages that are required to be supported.

with POSIX,
     POSIX_Sockets,
     POSIX_Sockets_Internet,
     POSIX_Report,
     POSIX_Process_Environment,
     System,
     POSIX_IO;

procedure p180402g is

   use POSIX,
       POSIX_Sockets,
       POSIX_Sockets_Internet,
       POSIX_IO,
       POSIX_Process_Environment,
       POSIX_Report;

   --  P180402 expects the following message
```

```
   Expected_Message : POSIX_String (1 .. 11) := "Send 4 test";

   procedure Send4_Tests (
      Socket   : in POSIX_IO.File_Descriptor;
      Buffer   : in System.Address;
      Options  : in Message_Option_Set := Empty_Set;
      To       : in Socket_Address_Pointer
               := Null_Socket_Address;
      Expected : in Error_Code := No_Error;
      Er1, Er2, Er3 : in String := "");

   procedure Send4_Tests (
      Socket   : in POSIX_IO.File_Descriptor;
      Buffer   : in System.Address;
      Options  : in Message_Option_Set := Empty_Set;
      To       : in Socket_Address_Pointer
               := Null_Socket_Address;
      Expected : in Error_Code := No_Error;
      Er1, Er2, Er3 : in String := "")
   is
      Sent    : POSIX.IO_Count;
   begin
      Send (Socket, Buffer, 11, Sent, To);
      if Er1 /= "" then
         Expect_Exception (Er1);
      else
         Comment ("Sent " & Integer'Image (Integer (Sent)) & " octets");
      end if;
   exception
   when E1 : POSIX_Error =>
      if Get_Error_Code /= Expected then
         Unexpected_Exception (E1, Er2);
      end if;
   when E2 : others => Unexpected_Exception (E2, Er3);
   end Send4_Tests;

----------------------------------------------------------------------------
--  Begin Tests

begin

   -------------------------------------------------------------------------
   --  Send on a socket

   Test ("Send 4 [18.4.13]");
   declare
      Socket  : POSIX_IO.File_Descriptor := 0;
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
      Port    : Internet_Port;
      Buffer  : POSIX_String (1 .. 11) := Expected_Message;
   begin
      Socket := Create (Internet_Protocol, Datagram_Socket, UDP);
      Port := Internet_Port (Integer'Value
        (To_String (Value (Argument_List, 3))));
      Set_Internet_Port (Int_Add, Port);
      Set_Internet_Address (Int_Add,
        String_To_Internet_Address ("127.0.0.1"));
      Send4_Tests (Socket, Buffer'Address, To => +Name);
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A001: p180402g");
   end;

   -------------------------------------------------------------------------
   --  Trying to send on a closed file descriptor generates
   --  the Bad_File_Desriptor error code.

   Test ("Send 4: Bad_File_Descriptor [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor := 0;
      Buffer : POSIX_String := "test";
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
   begin
      Set_Internet_Address (Int_Add,
        String_To_Internet_Address ("127.0.0.1"));
      Close (Socket);
      Send4_Tests (Socket, Buffer'Address, To => +Name,
                   Expected => Bad_File_Descriptor, Er1 => "A002: p180402g",
                   Er2 => "A003: p180402g", Er3 => "A004: p180402g");
   exception
      when E : others => Unexpected_Exception (E, "A005: p180402g");
   end;

   -------------------------------------------------------------------------
   --  Trying to send on a socket that is not connected generates
   --  the Not_Connected error code.

   Test ("Send 4: Not_Connected [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor;
      Buffer : POSIX_String := "test";
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
   begin
      Set_Internet_Address (Int_Add,
         String_To_Internet_Address ("127.0.0.1"));
      Socket := Create (Internet_Protocol, Stream_Socket, TCP);
      Send4_Tests (Socket, Buffer'Address, To => +Name,
                   Expected => Not_Connected, Er1 => "A006: p180402g",
                   Er2 => "A007: p180402g", Er3 => "A008: p180402g");
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A009: p180402g");
   end;

   -------------------------------------------------------------------------
   --  Trying to send on a file descriptor that is not a socket generates
   --  the Not_A_Socket error code.

   Test ("Send 4: Not_A_Socket [18.4.13.3]");
   declare
      Socket : POSIX_IO.File_Descriptor := 1;
      Buffer : POSIX_String := "test";
      Int_Add : aliased Internet_Socket_Address;
      Name    : Internet_Socket_Address_Pointer :=
        Int_Add'Unchecked_Access;
   begin
      Set_Internet_Address (Int_Add,
         String_To_Internet_Address ("127.0.0.1"));
      Send4_Tests (Socket, Buffer'Address, To => +Name,
                   Expected => Not_A_Socket, Er1 => "A010: p180402g",
                   Er2 => "A012: p180402g", Er3 => "A013: p180402g");
      Close (Socket);
   exception
      when E : others => Unexpected_Exception (E, "A014: p180402g");
   end;

exception when E : others => Fatal_Exception (E, "A017: p180402g");
end p180402g;
```