

IEEE Standards Interpretation for IEEE Std 1003.5™ (1999 Edition) - IEEE Standard for Information Technology--POSIX® Ada Language Interfaces--PART 1: Binding for System Application Program Interface

Copyright © 2005 by the Institute of Electrical and Electronics Engineers, Inc., 3 Park Avenue, New York, New York 10016-5997 USA. All Rights Reserved.

Interpretations are issued to explain and clarify the intent of a standard and **do not** constitute an alteration to the original standard. In addition, interpretations are not intended to supply consulting information. Permission is hereby granted to download and print one copy of this document. Individuals seeking permission to reproduce and/or distribute this document in its entirety or portions of this document must contact the IEEE Standards Department for the appropriate license. Use of the information contained in this document is at your own risk.

IEEE Standards Department, Copyrights and Permissions, 445 Hoes Lane, Piscataway, New Jersey 08855-1331, USA

December 2006

Interpretation Request #1

Topic: `Open_And_Map_Shared_Memory` and `Open_Or_Create_And_Map_Shared_Memory` functions **Relevant Clauses:** 12.5.1.2

This standard states that these functions are equivalent to a sequence of calls. This includes a call to `Open_Shared_Memory`, with the `Mode` parameter set as follows:

“If the value of `Protection` is set to `Allow_Write`, `Mode` is `Read_Write`; otherwise `Mode` is `Read_Only`.”

However, `Protection` is a `/set/` of options and may be set to “`Allow_Write + Allow_Read`”. By following the standard literally, this would lead to a `Mode` of `Read_Only`, which was clearly not what was intended.

This interpretation would make these functions unusable for opening or creating a shared memory mapping with both read and write access, which this user supposes is the most commonly desired mode of operation.

However, this user sees two possible interpretations that would make the functions usable:

1. The quoted sentence above should be interpreted as “If the value of `Protection` `/includes/` `Allow_Write`, `Mode` is `Read_Write`; otherwise `Mode` is `Read_Only`.”
2. As it is not specified exactly how the `Protection` parameter should be used in the sub-

sequent call to `Map_Memory`, a possible interpretation would be that one should always add the `Allow_Read` option in the call to `Map_Memory`.

In this user's opinion, number #2 (above) is counterintuitive, as one would expect the protection parameters passed on to `Map_Memory` being the same as those initially submitted. And besides that, this would decrease the flexibility by not allowing the user to create a write-only mapping.

To illustrate the problematics, a simple example will follow. The example consists of two Ada functions which communicate via a shared memory area. With a common implementation of the IEEE 1003.5 (Florist), the problem arises in the form of a `Storage_Error` when trying to access the shared area:

```
---[shmserv.adb]--- with POSIX_Generic_Shared_Memory, POSIX_IO, POSIX_Memory_
Mapping, POSIX.Permissions; with Ada.Text_IO; procedure Shmserv is type Data is
record I : Integer; J : Integer; end record; package GSM is new Posix_Generic_Shared_
Memory(Data); Descriptor : POSIX_IO.File_Descriptor; Data_Access : GSM.Shared_Ac-
cess; begin Descriptor := GSM.Open_Or_Create_And_Map_Shared_Memory ("/foo",
POSIX_Memory_Mapping.Allow_Write, POSIX.Permissions.Access_Permission_Set);
Data_Access := GSM.Access_Shared_Memory(Descriptor); Data_Access.I := 1; loop
Ada.Text_IO.Put_Line(Integer'Image(Data_Access.I)); delay(0.5); end loop; end Shm-
serv; ---[end shmserv.adb]---
```

```
---[shmcli.adb]--- with POSIX_Generic_Shared_Memory, POSIX_IO, POSIX_Memo-
ry_Mapping; procedure Shmcli is type Data is record I : Integer; J : Integer; end record;
package GSM is new Posix_Generic_Shared_Memory(Data); Descriptor : POSIX_IO.
File_Descriptor; Data_Access : GSM.Shared_Access; begin Descriptor := GSM.Open_
And_Map_Shared_Memory ("/foo", POSIX_Memory_Mapping.Allow_Write); Data_Access
:= GSM.Access_Shared_Memory(Descriptor); loop Data_Access.I := Data_Access.I + 1;
delay(1.0); end loop; end Shmcli; ---[end shmcli.adb]---
```

The current Florist implementation of `Open_And_Map_Shared_Memory` is defined as follows:

```
---- function Open_And_Map_Shared_Memory (Name : POSIX.POSIX_String; Protection
: POSIX.Memory_Mapping.Protection_Options; Masked_Signals : POSIX.Signal_Masking
:= POSIX.RTS_Signals) return POSIX.IO.File_Descriptor is FD : POSIX.IO.File_Descrip-
tor; Mode : POSIX.IO.File_Mode; begin if Protection = POSIX.Memory_Mapping.Al-
low_Write then Mode := POSIX.IO.Read_Write; else Mode := POSIX.IO.Read_Only; end
if; Begin_Critical_Section; begin FD := POSIX.Shared_Memory_Objects.Open_Shared_
Memory (Name, Mode, POSIX.IO.Empty_Set, Masked_Signals); POSIX.IO.Truncate_File
(FD, Length); Insert_Node (FD, POSIX.Memory_Mapping.Map_Memory (System.Stor-
age_Elements.Storage_Offset (Length), Protection, POSIX.Memory_Mapping.Map_
Shared, FD, 0)); End_Critical_Section; exception when others => End_Critical_Section;
raise; end; return FD; end Open_And_Map_Shared_Memory; ----
```

Interpretation Response

There is a problem with the wording of 12.5.1.2, and that your suggested interpretation (item 1 above) is correct. That is where the standard says:

“If the value of Protection is set to Allow_Write, Mode is Read_Write; otherwise Mode is Read_Only to convey the intended meaning it should instead say

“If the value of Protection **includes** Allow_Write, Mode is Read_Write; otherwise Mode is Read_Only”.

Accordingly, it seems the “=” test in the Florist implementation code below should be changed to “>=”.

```
if Protection = POSIX.Memory_Mapping.Allow_Write then Mode := POSIX.IO.Read_
Write; else Mode := POSIX.IO.Read_Only; end if;
```