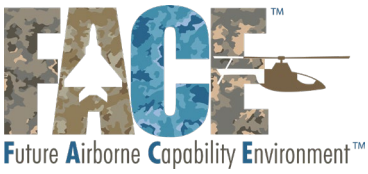

FACE™ Conformance Verification for Ada Software

Issues and Proposed Approach for non-OSS Ada UoCs

The Open Group FACE™ and SOSA™ US Air Force TIM Paper by:

Benjamin M. Brosgol. Albert Lee, Patrick Rogers, and Dudrey Smith
AdaCore
{brosgol, lee, rogers, smith}@adacore.com

September, 2022

Table of Contents

Executive Summary

The current procedures for FACE conformance verification are based on linking Unit of Conformance (UoC) object code with standard libraries – "Gold Standard Libraries", or GSLs – and checking for unresolved references. Although this approach works well for UoCs written in C or C++, it is not adequate for Ada. (It also does not fit the Java model, but Java-related issues are outside the scope of this paper.) In Ada source code, run-time functionality is realized in a portable way not by invoking a standard library but rather by using standard language syntax. The library Application Program Interface (API) supporting Ada run-time functionality is invoked from the compiled code, not from the source program, and is vendor specific.

This paper presents a proposal for FACE conformance verification of Ada UoCs based on the concept of a *stubbed run-time library*, comprising a selected set of package specs and their associated "dummy" package bodies, to be included as a supplement to the Ada GSL for the targeted profile and capability set in the Conformance Test Suite (CTS). The stubbed run-time library is specific to the Ada compiler that is used by the CTS. It will support the features permitted in its targeted capability set and exclude, to the extent possible and practical, the features that the capability set prohibits. The proposed approach extends the current CTS verification procedures and is under consideration in the FACE Consortium.

A large Ada codebase of defense airborne software has been developed and deployed over the years, and a practical and effective conformance policy for Ada can help expand the FACE ecosystem while also making the FACE approach attractive to Ada developers.

The proposed approach detailed in this paper covers both Ada 95 and Ada 2012. It applies to UoCs outside the Operating System Segment (OSS) and is currently focused on verification of Ada UoCs in the Portable Components Segment (PCS), but the basic concepts also apply to UoCs in the other non-OSS segments.

Introduction

As defined in FACE Consortium standards [1, 2, 3, 4], conformance verification for UoCs in the PCS entails meeting several kinds of requirements:

- Conformant USM (Unit of Portability Supplied Data Model),
- Conformant uses of the Transport Services Segment (TSS),
- Adherence to the restrictions specified in the targeted profile, and
- Adherence to the restrictions specified in the targeted capability set.

The requirements associated with conformant TSS usage and data model do not raise Ada-specific issues and will not be dealt with in this paper. Likewise, the profile restrictions (POSIX and ARINC 653 subsets) can be handled for Ada in the same way as for C and C++, by including the permitted functions in the GSL. On the other hand, enforcing adherence to the capability set restrictions presents an issue with Ada that does not arise as a major problem in C or C++. For the latter languages[1], run-time functionalities such as threading, memory management, and exception handling are implemented by a standard library API. A GSL for a specific capability set can simply include the API subset permitted by that capability set. If a candidate UoC invokes a function outside of that subset, the error is detected by the CTS; the object code for the UoC will not link with the GSL, since there will be an unresolved reference.

For Ada, however, there is not a standard API for the run-time services defined by the language. Portability for an Ada program using threading (known in Ada as "tasking"), memory management, or exception handling is achieved not through source code invocation of standard library API functions as in C but rather through standard language syntax. The compiled code contains calls on an API that implements the run-time functionality, but this API (the run-time library), in both its interface and its implementation, is specific to the Ada compiler vendor. As an example, here is a skeletal version of a simple Ada package:

[1] Some C++ features result in the invocation of compiler specific functions. However, for C++ the names of these functions are well known for the GCC compiler. Further, in many cases these functions are defined as part of the platform's application binary interface (ABI) and are even more well known. These are in the "allowed definitions" files in the CTS. In Ada this problem in more severe, since the names are completely compiler specific. [9]

```
package Pkg is
   task T1;
end Pkg;

with Interfaces.C;
package body Pkg is
   task body T1 is
      N : Interfaces.C.int;
   begin
      …
   end T1;
end Pkg;
```

When this code is compiled using AdaCore's GNAT Ada implementation, the object code will contain references to external symbols including the following:

- `Interfaces.C`, a package from the Ada standard library. This package is explicitly referenced in the source code.
- `System.Tasking`, a package in AdaCore's run-time library that implements task management (creation, queuing, etc.). This package is not referenced from the source code but rather from the compiled object code. If the Ada source code is compiled with a different vendor's compiler, a differently named package would almost certainly be referenced.

The above Ada code conforms with each of the Ada FACE profiles and capability sets (for both Ada 95 [5] and Ada 2012 [6]) provided that the code within "..." in the body of the task `T1` obeys the profile and capability set restrictions. However, since run-time library packages that implement language functionality are not part of the language standard, the CTS procedures for generating GSLs do not take them into account. The result, if the CTS performed an analysis on Ada in the same way as it is done for C and C++, would be a failure result for the above code, even though it is FACE conformant[2].

The issues associated with run-time libraries have been known for some time, and the conformance verification procedures presently support two approaches:

- Treat the run-time library as part of the Operating System Segment (OSS)

  If the Real-Time Operating System (RTOS) has been verified as FACE conformant, then the run-time library is linked with the UoC object code to test for conformance. All references to the library are resolved, and the external references from the library implementation are "bottom side" interfaces that are allowed.

- Treat the run-time library as part of the UoC

[2] Actually, in the current CTS, Ada is not handled at all. The Conformance Verification Matrix (CVM) shows each Ada capability set requirement as "Verification not needed".

With this approach, the run-time library is subject to the same conformance requirements as the UoC application code.

The first approach avoids the issue of justifying "bottom side" interfaces, but it requires the Ada UoC to be targeted to an RTOS that includes the relevant run-time libraries and has been verified as FACE conformant. However, deployment of FACE conformant UoCs in multiple kinds of environments, including OSes that are not FACE conformant, is important in attracting avionics software developers to adopt the FACE approach. Depending on the run-time library to be part of a conformant RTOS is overly constraining. Additionally, even with a FACE conformant RTOS, the capability set restrictions must be enforced. For each of the three capability sets, a specific run-time library will be needed.

The alternative approach, treating the Ada run-time library as part of the UoC, has proved to be unwieldy in practice. Explaining and justifying the "bottom-side" calls from the library implementation is a major effort. It assumes a level of expertise with compiler technology that is possessed by the compiler vendor but not necessarily by the UoC developer or the Verification Authority representative. Moreover, from a FACE perspective there is no reason why the implementation of the run-time library should obey the same restrictions as the UoC itself. By its nature a run-time library is target dependent and non-portable. It is the "glue" that connects the FACE conformant UoC to the target platform and that enables the UoC source code to be portable.

Furthermore [10], most Ada run-time libraries have years or even decades of flight history experience, including certification under assurance standards such as DO-178B [7] and DO-178C [8], and were not designed to meet the requirements of the FACE profiles and capability sets. It would be financially burdensome to Ada compiler vendors, and counterproductive to the FACE community, for Ada vendors to rework their existing solutions. Portability is needed for the UoC (via its use of standard APIs and language features), not for the run-time library that implements these features.

The following sections of this paper propose an alternative approach, which can be accomplished in the context of the current CTS. Through the concept of a "stubbed run-time library" that is included when an Ada GSL is generated, the test procedures can be extended to support verification of Ada UoCs. The approach applies to all profiles and capability sets and to both Ada 95 and Ada 2012.

Host-Based Conformance Verification

The CTS supports two styles of conformance verification:

- Host-based, on a Windows or Linux platform
- Target-based, using the UoC's cross-development toolchain (discussed in the next section)

For host-based conformance verification the CTS supports both Windows 10 and Linux. For Windows 10, the installation process sets up an Ada compiler derived from a Free Software Foundation version of GNAT. For Linux, the CTS supports Red Hat Enterprise Linux versions 7 and 8 (RHEL 7 and RHEL 8). RHEL 7 supplies an Ada compiler that handles Ada 95 but not Ada 2012. RHEL 8 does not include an Ada compiler.

Based on FACE Consortium policy, language support in the CTS requires a non-proprietary compiler. To meet that requirement, the GNAT GPL 2017 Ada compiler is proposed. This edition is publicly available with licensing that is appropriate for usage in the CTS. It implements both Ada 95 and Ada 2012 and thus allows a single compiler to be used for both versions of the language. (Later versions of the GNAT GPL technology[3] only provide Ada 2012; verifying Ada 95 UoCs would require supplying GNAT GPL 2017 as a second Ada toolchain,) Since RHEL 7 is approaching its end-of-life stage, RHEL 8 is proposed as the Linux edition for hosting this Ada compiler. GNAT GPL 2017 is available for Windows 10.

All GNAT compilers implement the full Ada language. For use in the CTS the implementation for a specific capability set needs to allow all features that are permitted by that capability set, and it should prohibit, to the extent possible and practical, features that the capability set excludes. To this end AdaCore has prepared three run-time libraries[4], corresponding to the three capability sets:

- The General-Purpose run-time library was derived from the full Ada run-time for the GNAT GPL 2017 compiler by removing units that are prohibited in the General-Purpose capability set.
- The Safety-Extended run-time library was derived from the General-Purpose run-time library by removing units that are prohibited in the Safety-Extended capability set.
- The Safety-Base & Security run-time library was derived from the Safety-Extended run-time library by removing units that are prohibited in the Safety-Base & Security capability set.

Each of these run-time libraries can be used for both Ada 95 and Ada 2012. For Ada 95 UoCs a compiler switch `-gnat95` will enforce the restriction to Ada 95 features (usage of new Ada 2012 features are then flagged as errors).

[3] In 2019 the GNAT GPL edition was rebranded as GNAT Community.
[4] These subsetted libraries were prepared by manually editing the source code. As a more efficient long-term approach, AdaCore is also working on an alternative and automated scheme based on removing symbols from the library object file.

For use in the CTS, there is no need for the run-time libraries to include their full (functional) implementation. Instead, each will consist of a stubbed version: a package specification sufficient to provide the permitted functionality, and an "empty" package body (i.e., with null bodies for the implementation of the various subprograms). This is analogous to what is done for C and C++: header (`.h`) files and stubbed `.c` and `.cpp` files.

The stubbed Ada run-time libraries do not include the POSIX or ARINC 653 APIs that are specified for the various profiles. These APIs should be included in the Gold Standard Libraries for Ada as part of the CTS GSL generation process for Ada.

The stubbed Ada run-time libraries are specific to a particular version of the FACE Technical Standard. If a new version of the Technical Standard makes any modifications to the Ada capability sets, for example by permitting a feature that has previously been prohibited, then the corresponding stubbed run-time library will need to be adapted accordingly.

The three stubbed run-time libraries are not an exact match for the restrictions in the capability sets; i.e., there are restrictions that are not enforced by the library. This is due to several reasons:

- Some restrictions are syntactic in nature; for example, the rule that prohibits implementation-defined pragmas from being applied to data structures from FACE interfaces. Such restrictions are not enforceable by link-time checks and instead require analysis and/or source code inspection. This issue applies not only to Ada but also to C and C++ (and Java).
- Some restrictions would require retooling the compiler, resulting in a GNAT version that is not the same as the GNAT GPL 2017 edition. An example is the prohibition of `Wide_Character` and `Wide_String`. The declarations of these types are in the special predefined package `Standard`, and changing `Standard` would require a variant version of the GNAT GPL 2017 compiler. Instead, usages of these types can be detected by appropriate instances of pragma `Restrictions`[5] in the source code of the UoC. In the presence of such pragmas, uses of these types are in error and will result in compile-time failures. An object file presented to the CTS for linkage will never contain a reference to the prohibited types.

Capability set violations thus fall into two categories:

- *Usage of a prohibited feature that is excluded from the stubbed library*. In almost all cases such violations will result in compile-time errors, since the compiler will fail to compile a program that uses a feature that is not included in its associated run-time. For readability the UoC developer may want to include relevant instances of pragma `Restrictions`, to document the intent to exclude such features. In rare cases a feature that is excluded from the stubbed library will be permitted by the compiler (for example, interfacing to a prohibited POSIX C function). In such a

[5] Pragma `Restrictions` is a language-defined pragma that allows a project to self-impose restrictions in the form of prohibitions (such as `No_Task_Allocators`) or limits (`Max_Entry_Queue_Length`). The Ravenscar subset of the Ada tasking model is defined by a set of Restrictions pragmas.

situation the violation will be caught by the link-time tests in the CTS.

- Usage of a prohibited feature that is not excluded from the stubbed library. There are two sub-cases:
- *Use of the feature can be detected by the compiler*. An example is `Wide_Character` and Wide_String as noted above. Pragma `Restrictions` is needed to detect the violation[6]. The CTS documentation will identify the full list of pragmas that are needed for each capability set (AdaCore will furnish this information), and the UoC developer will need to use these pragmas when generating the object module used in the CTS and demonstrate to the Verification Authority that the pragmas were used during compilation.
- *Use of the feature is not detectable by the compiler*. An example is the use of an implementation-defined pragma on a FACE interface data structure. Analysis and/or source code inspection is then needed. As the supplier of the GNAT GPL Ada compiler and its associated stubbed run-time libraries, AdaCore will document all such features. A supplemental static analysis tool can assist in automating this step.

The Conformance Verification Matrix (CVM) will identify, for each verifiable requirement, how the verification will be conducted. As is done now, it will either be through the CTS (link-time tests) or inspection. The following table, based on the CVM for the FACE Technical Standard 3.1, illustrates what this would look like for some of the requirements in the Safety Extended Capability Set for Ada 2012. The entry for Verification Method is marked as "Analysis / Inspection" rather than simply "Inspection", since the UoC developer may find it helpful to use an automated tool.

[6] In this case `pragma Restrictions(No_Wide_Characters)` will report as errors any uses of `Wide_Character`, `Wide_String`, `Wide_Wide_Character`, or `Wide_Wide_String`.

Verification of Requirements for Ada 2012 Safety-Extended Capability Set				
Row ID	**Verification Needed**	**Requirement Summary**	**Verification Method**	**Note**
F-433	Y	No use of implementation-defined pragmas on FACE interface data structures	Analysis / Inspection	
F-437	Y	No Asynchronous Transfer of Control	Test	
F-438	Y	No use of synchronized, task or protected interfaces	Analysis / Inspection	
F-439	Y	No use of Exception_Information or Exception_Message functions	Test	
F-440	Y	No use of certain forms of generic formal packages	Analysis / Inspection	
F-441	Y	No use of deallocation	Test	
F-442	Y	No use of Wide_Character, Wide_String	Test	Need pragma Restrictions (No_Wide_Characters)
F-443	Y	No use of Wide_Wide_Character, Wide_Wide_String	Test	Need pragma Restrictions (No_Wide_Characters)
F-444	Y	No use of random number generation	Test	
F-445	Y	No use of input-output	Test	
F-446	Y	No use of containers library	Test	
F-447	Y	No use of Distributed Systems Annex	Test	
F-448	Y	No use of Information Systems Annex	Test	
F-449	Y	No use of Numerics Annex	Test	
F-550	Y	No use of unbounded strings	Test	

Target-Based Conformance Verification

The approach to target-based verification is similar to host-based. The provider of the full Ada run-time library (either Ada 95, Ada 2012, or both) that is part of the cross-compilation toolchain needs to supply a subsetted stubbed library for each of the three capability sets. The Ada GSL for each capability set will be supplemented by the stubbed library for that capability set. The provider of these libraries may need to adapt entries in the CVM, for example if some requirement uses link-time tests for host-based verification but code inspection for target-based verification, or vice versa).

Vetting an Ada Stubbed Run-Time Library

A stubbed run-time library that is used to verify conformance of an Ada UoC needs to supply the functionality needed by that UoC. For the stubbed libraries included in the CTS for use during host-based verification, this means that each library must implement at least the features that are permitted by the capability set with which it is associated. The library provider can demonstrate this property by using a set of test cases that exercises the permitted features and by showing successful compilation of these tests. Target-based verification does not need to carry out this step, since in such a situation the only requirement is that the specific candidate UoC is successfully compiled.

As noted earlier, it is not possible for a stubbed run-time library to prohibit all features that are excluded from its associated capability set. Nevertheless, a quality aspect of the library is how closely it approximates this ideal (i.e., to minimize the number of excluded features that are allowed by the library). To this end, the library provider should use a set of tests exercising each of the language features prohibited from the corresponding capability set, and show that these result in compilation failures. (In this context, a compilation failure demonstrates successful rejection of an excluded feature.) In any event, the library provider needs to document each prohibited feature whose support is not excluded from the stubbed library, since this affects the verification method: either compilation of the UoC with a relevant pragma `Restrictions` is required, or else source inspection / static analysis will be needed.

The implementation of the stubbed library is not relevant to conformance verification for the UoCs. Although the run-time library enables Ada source code portability, its implementation is not itself portable. Its API is compiler dependent (e.g., a stubbed library for GNAT will not work with a Green Hills Ada compiler, and vice versa), and its implementation invokes OS- and platform-specific services. In particular, the library implementation does not need to adhere to the restrictions in the targeted profile and capability set:

- The Linux host OS is itself not FACE conformant, and the stubbed run-time library can be regarded as part of the host platform.
- There are no such requirements on the implementation of the GSL functions that are used for C.

Summary

The current FACE conformance procedures do not handle non-OSS Ada UoCs. To address this issue, we have proposed an approach that involves stubbed run-time libraries. More specifically, for host-based conformance verification:

- The CTS should use the publicly available GNAT GPL 2017 edition on Red Hat Enterprise Linux 8 and on Windows 10. AdaCore can make available to the FACE Consortium this version of GNAT, which supports both Ada 95 and Ada 2012. GNAT GPL 2017 has licensing appropriate for use in the CTS, both by the Verification Authority and by software suppliers.
- For each Ada capability set, the CTS for host-based conformance verification should generate a Gold Standard Library containing a stubbed version of subset of the standard Ada library supporting the permitted features, supplemented by a stubbed Ada run-time library that supports at least the features allowed by that capability set, and that excludes, to the extent practical, the features that are prohibited by that capability set. The Ada GSL should also include the C GSL functions for POSIX or ARINC 653 as appropriate, to facilitate verification of Ada UoCs that explicitly interface to these functions.
- The supplier of the stubbed libraries (AdaCore) should show that the allowed features are supported, and how comprehensively the prohibited features are excluded, through an appropriate set of test cases. Tests of the permitted features should be compiled successfully, and tests of prohibited features should result in compilation errors. Since the latter might not always be possible, prohibited features that are not excluded need to be documented by the library provider. These steps need to be taken by the library provider (and demonstrated to the Verification Authority) before the libraries are incorporated into the CTS.
- The full implementation of the stubbed libraries is irrelevant as far as usage in the CTS is concerned.
- The CVM needs to be adapted to indicate the verification method (link-time test, or inspection / analysis) for each requirement

Target-based conformance verification is similar in terms of the derivation of stubbed run-time libraries by the library provider, but there is no need to demonstrate that the libraries support all features permitted by the capability sets.

These proposed modifications to the conformance test procedures are evolutionary and in the spirit of the existing methodology. They do not require access to the UoC source code, and they continue to rely on link-time testing to ensure that there are no unresolved external references. The main difference between Ada and C or C++ is that in Ada, almost all violations of capability set restrictions will be caught at compile time, because of the language's early error detection. As a result, the link-time tests will mainly be a confirmation of the already known conformance with the FACE requirements.

Although the PCS is likely to be where Ada will mainly be used, for completeness the verification procedures should also cover Ada UoCs in the other segments of the FACE reference architecture. This is planned as a future effort.

References

(Please note that the links below are good at the time of writing but cannot be guaranteed for the future.)

[1] *FACE Conformance Certification Guide, Edition 1,0* (x1606), published by The Open Group, August 2016; https://publications.opengroup.org/certifications/face/x1606

[2] *FACE Conformance Test Suite(s)* published by The Open Group; https://www.opengroup.org/face/conformance-testsuites

[3] *FACE™ Conformance Verification Matrix, Edition 3.1* (x205), published by The Open Group, February 2021; publications.opengroup.org/certifications/face/x205

[4] *FACE™ Technical Standard, Edition 3.1* (C207), published by The Open Group, July 2020; www.opengroup.org/library/c207

[5] *Ada Reference Manual ISO/IEC 8652:1995(E) with Technical Corrigendum 1, Language and Standard Libraries* https://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-TTL.html

[6] *Ada Reference Manual ISO/IEC 8652:2012(E) with Technical Corrigendum 1, Language and Standard Libraries* http://www.ada-auth.org/standards/rm12_w_tc1/html/RM-TTL.html

[7] *RTCA /EUROCAE DO-178B/ED-12B – Software Considerations in Airborne Systems and Equipment Certification*, December 1992.

[8] *RTCA /EUROCAE DO-178C/ED-12C – Software Considerations in Airborne Systems and Equipment Certification*, December 2011.

[9] Joel Sherrill, referee comment on TIM paper submission; July 2022.

[10] Joel Sherrill, referee comment on TIM paper submission, July 2022

About the Authors

Dr. Benjamin Brosgol is a member of AdaCore's senior technical staff. He has been involved with programming language design and implementation throughout his career, concentrating on languages and software engineering technologies for high-assurance systems. He was a member of the design team for Ada 95, and he also served in the Expert Group for the Real-Time Specification for Java (Java Specification Request JSR-001). Dr. Brosgol is an active member of the FACE Technical Working Group's (TWG) Operating Systems Subcommittee as well as the Enterprise Architecture Airworthiness subteam, and he served as Vice Chair of the TWG between June 2020 and June 2022. He has written papers and delivered talks at several FACE TIMs.

Mr. Albert Lee is a senior software engineer at AdaCore. He has experience as a developer, support technician, field application engineer, and language and tools trainer, and has extensive experience working with customers to develop applications for native and embedded systems using AdaCore's tools. Mr. Lee contributed to the design of the Ada conformance verification proposal presented in this paper, and he developed the proof of concept demonstrating the feasibility of using stubbed Ada run-time libraries with the FACE CTS.

Dr. Patrick Rogers has been a computing professional since 1975, primarily working on microprocessor-based real-time applications. He began working with Ada in 1980 and was director of the Ada9X Laboratory for the U.S. Air Force's Joint Advanced Strike Technology Program, and Principle Investigator in distributed systems and fault tolerance research projects for the U.S. Air Force and Army. As a member of the senior technical staff at AdaCore, he is a developer of the bare-board products for Ada and provides support and training focused on embedded real-time applications. Dr. Rogers is the principal engineer responsible for deriving the stubbed Ada run-time libraries described in this paper.

Dr. Dudrey Smith is a senior embedded system development consultant at AdaCore. He has been involved with military and commercial embedded system/software development and certification for more than 40 years, with major leadership roles at companies including Lear Siegler, Smiths Aerospace, and General Electric Aviation Systems. Dr. Smith is an active member of the FACE Operating Systems and Conformance Subcommittees.

About The Open Group FACE™ Consortium

The Open Group Future Airborne Capability Environment™ Consortium (the FACE™ Consortium), was formed as a government and industry partnership to define an open avionics environment for all military airborne platform types. Today, it is an aviation-focused professional group made up of industry suppliers, customers, academia, and users. The FACE Consortium provides a vendor-neutral forum for industry and government to work together to develop and consolidate the open standards, best practices, guidance documents, and business strategy necessary for acquisition of affordable software systems that promote innovation and rapid integration of portable capabilities across global defense programs.

Further information on the FACE Consortium is available at www.opengroup.org/face.

About The Open Group SOSA™ Consortium

The Open Group SOSA™ Consortium enables government and industry to collaboratively develop open standards and best practices to enable, enhance, and accelerate the deployment of affordable, capable, interoperable sensor systems. The SOSA Consortium is creating open system reference architectures applicable to military and commercial sensor systems and a business model that balances stakeholder interests. The architectures employ modular design and use widely supported, consensus-based, nonproprietary standards for key interfaces.

Further information on the SOSA Consortium is available at www.opengroup.org/sosa.

About The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through technology standards. With more than 870 member organizations, we have a diverse membership that spans all sectors of the technology community – customers, systems and solutions suppliers, tool vendors, integrators and consultants, as well as academics and researchers.

The mission of The Open Group is to drive the creation of Boundaryless Information Flow™ achieved by:

- Working with customers to capture, understand, and address current and emerging requirements, establish policies, and share best practices
- Working with suppliers, consortia, and standards bodies to develop consensus and facilitate interoperability, to evolve and integrate specifications and open source technologies
- Offering a comprehensive set of services to enhance the operational efficiency of consortia
- Developing and operating the industry's premier certification service and encouraging procurement of certified products

Further information on The Open Group is available at www.opengroup.org.