



# SOFTWARE PORTABILITY

**Alan Bell**

**TESSELLA SUPPORT SERVICES PLC**

Issue V1.R2.M1

July 1998



## SOFTWARE PORTABILITY

### Introduction

Portability is one of the most important issues facing application developers today. End users are demanding the same functionality on the whole spectrum of systems available to them. It is in the interests of the developer to recognise this as early as possible and to engineer the application with this in mind.

Portability does not mean the same thing as *porting* or *migration*. Migration involves simply making an existing application run successfully on a new platform and can often result in replacing one set of system dependencies with another. Each port may be a non-trivial task, and the lessons learned in one port may not be relevant to the next. The term *portable* implies that the software was intended for several platforms from the beginning and that this factor was considered throughout the design and implementation. It is only by taking this approach that one can ensure a high degree of portability in the finished application.

When one mentions portable software the association is inevitably made with Open Systems. It will be useful to keep in mind the IEEE definition of an Open System given in IEEE P1003.0 “Guide to POSIX Open System Environment”

***Open system:*** *A system that implements sufficient open specifications for interfaces, services and supporting formats to enable properly engineered applications software:*

- to be ported across a wide range of systems (with minimal changes)*
- to interoperate with other applications on local or remote systems*
- to interact with users in a style which facilitates user portability*

Thus portability does not imply that the application will simply require to be recompiled for the new platform. Neither does it allow for major re-engineering projects. Portability is the ability to provide the application on another system with known and economically reasonable cost and effort.

There are several dimensions to application portability:

- Program Portability**

This is the aspect that one associates with software portability, namely will the code run successfully on all intended platforms.
- Data Portability**

Are the data structures used in the application's data files available on all the platforms?
- End-User Portability**

Since re-training of users can be a relatively expensive exercise, it is often required that an application will have the same look and feel across several platforms. At the opposite extreme, the application may require a Motif graphical User Interface on some platforms and the familiar Microsoft Windows interface on PCs. This flexibility must be designed in from the outset.
- Developer Portability**

The use of a standard set of interfaces and services on all the target platforms minimises the re-training of developers required when a new platform is added to the list.
- Documentation Portability**

Users of applications on different platforms often have different expectations regarding the type of documentation they receive especially in the context of on-line help facilities.

Here we will mainly be concerned with the first of the above considerations. However, there will frequently be significant overlap with the other aspects.

### **Profiles**

One approach to ensuring portability is to use a standard set of application programming interfaces (API) and services which are available on all the target platforms. Such a common set of services is known as a profile. This profile may be defined by one of the standards bodies or may be an internal profile defined by a business and to which all internal software must adhere. The choice of profile may depend on several factors. Some customers may specify a particular profile. The cross-platform consistency of, for example, using the X Windows toolkit library must be balanced against the time and effort savings to

be obtained by using Motif which implicitly defines the look and feel of the user interface.

<b>Defining Body</b>	X/OPEN CAE (base XPG 5)	OSF AES	NIST APP (USA)
<b>Operating System</b>	POSIX Internationalisation	POSIX and X/Open(BASE)	POSIX
<b>Languages</b>	C, Fortran, Pascal, A, COBOL	C, Fortran, Pascal, ADA, COBOL, LISP, BASIC	C, Fortran, Pascal, ADA, COBOL
<b>User Interface</b>	X Windows	X Windows OSF/MOTIF	X Windows
<b>Data Management</b>	ISAM, SQL	SQL	SQL, IRDS
<b>Data Interchange</b>			IGES, SGML, ODA/ODIF
<b>Network Services</b>	XTI	OSI, OSF/DCE TCP/IP	OSI
<b>Graphics</b>	N/A	GKS/PHIGS	GKS, PHIGS

#### **Standard profiles for application development**

Applications developed using only the elements of the specified profile will be portable across all platforms on which the profile is available. It should be stressed that the use of profiles depends on strict adherence to the defined standard for any language, interface or service used. It is pointless to say 'I am using the OSF profile' and then to use platform dependent Fortran extensions and the extensions supplied with your particular PHIGS implementation. The final application may well turn out to be very difficult to port to another platform which, although it supports all the elements of the OSF profile, does not provide the same extensions.

There will always be cases in which it is difficult or even impossible to stick rigidly to a profile or a standard. Performance issues or a requirement for extra functionality may make it necessary to use non-standard extensions to your chosen profile. The following sections will outline some methods which can be used to aid portability in these cases.

## Operating System Dependencies

The POSIX standard ( IEEE standard 1003.1-1988 , Portable Operating System Interface for Computer Environments) specifies a common set of interfaces to all operating systems which are POSIX compliant , such as `system()` which allows operating system calls to be made from a high level language. For C programs, the POSIX interface is accessed implicitly through the standard input/output library, `stdio`.

Since the POSIX interface is itself designed to be portable, it is unlikely that it will provide all of the functions available on a particular platform. If an application needs access to operating system features to which POSIX does not provide an interface then the developer has two options if portability is to be maintained.

1. Use another portable interface to implement the feature.
2. Use the Abstraction and Isolation techniques described in this booklet.

## Hardware Dependencies

Hardware dependencies are probably the hardest to avoid when designing portable software. Often, the code contains no statements which can be labelled as machine specific. Hardware dependencies tend to stem from assumptions made during the design and implementation of the application. Examples of possible hardware dependencies include:

- The order of bits and bytes in memory.
- The relative sizes of integers and addresses
- The maximum and minimum values of integers
- The representation of floating point data
- The maximum available address

It is possible to minimise the risk of these types of hardware dependencies appearing unnoticed by following a few good coding practices:

- Explicitly declare the data types of all functions and variables. Declare, for example, an integer as either long or short. The default length of type integer is system dependent.
- Avoid using mixed mode expressions. Explicitly cast variables to a consistent type.

- ❑ Use variables for one purpose only. For example, do not use a variable as both a pointer and an integer.

### **Abstraction and Isolation**

Abstraction and isolation are two methods used by developers to maintain a high degree of portability in an application in which there are unavoidable system dependencies. Abstraction involves structuring the application code around a conceptual model of the processes which must be performed rather than concentrating solely on the logical operations.

The simplest and most intuitive ‘abstraction mechanism’ is procedure calling. Procedures help the developer to focus on the various tasks performed in a section of code without being swamped by the details of how these tasks are actually performed.

Procedural abstraction can be used along with isolation to separate system dependencies from the rest of the application. If, rather than calling a system dependent function, the application calls a generic interface which in turn calls the system dependent function then this layer of abstraction makes it easier to deal with the dependency when the application is ported.

Take the example of a Graphical User Interface. Although it is possible to provide a Motif interface across a wide range of platforms including personal computers, it may be necessary to provide a Microsoft Windows interface because PC users feel more comfortable with an application which looks and feels like their other applications. If the application and the GUI are closely linked, with calls to the Motif API throughout the code, then the portability of the application to MS Windows may be questionable.

If, on the other hand, all calls in the body of the application are to a generic interface which in turn makes the calls to the appropriate API, then the modules containing the generic interface routines can be labelled as potentially platform dependent. When a new API is used, a new set of routines can simply be substituted. The dependencies have been abstracted from the application and isolated in a set of platform dependent routines.

The abstract and isolate technique can also be used with data abstraction. In data abstraction, data is described, manipulated and accessed only in terms of a set of operations performed on that data. As with procedural abstraction, the portability benefit comes from the knowledge that even if the implementation of the

abstraction changes, the interface will remain the same. Data abstraction hides the details of how and where the data is stored because, like the details of a procedure, it is not crucial to the use of the data.

As an example let us consider a queue. The application needs to be able to create a new queue, add an item to a queue, get the next item in a queue and find out how many items are in a particular queue. The developer does not need to know whether the queue is actually a linked list or a more complicated structure. All that is important is that there exist four routines:

- Queue\_Create(qname) that creates a new queue with name qname
- Queue\_Put(qname,item) that adds item to the end of the queue qname
- Queue\_Get(qname,item) that fetches item from the front of queue qname
- Queue\_HowMany(qname) that returns the number of items in the specified queue.

These routines may contain platform dependent calls to, for example, high performance file handling routines. The abstraction allows the routines to be isolated and easily dealt with during a port.

## Conclusions

Portability is almost always a matter of compromise. Portability must be balanced against, for example, integration with other, possibly proprietary software on a particular platform. Several degrees of portability can be defined:

- Generic Portability* where the application is written entirely in standard high level languages.
- Profile Based Portability* as described earlier.
- Operating System Portability* which may include both of the above but the application uses some operating system specific functions.
- Minimal Portability* in which the application utilises some services specific to a particular hardware platform.

It is always preferable for an application to fall into one of the first two categories if multiple platforms are to be supported. However, if portability is a consideration throughout the development process, even an application which falls under the heading of minimal portability, but which uses abstraction and isolation in the necessary places, may still retain a high degree of portability.

**Tessella Support Services plc**  
**Creating Software for Science and Engineering**

Tessella's services range from feasibility studies, through system design, development, implementation and ongoing support. Our expertise includes:

Data Analysis Software  
Data Capture  
Simulation Software  
Advanced Graphics  
Systems Support  
Database Applications

**Other Technical Supplements available include:**

- |   |  |
|---|--|
| <input type="checkbox"/> Archiving of Electronic Info   | <input type="checkbox"/> Object Oriented Programming   |
| <input type="checkbox"/> Active Server Pages            | <input type="checkbox"/> Pocket PC                     |
| <input type="checkbox"/> Automated GUI Testing          | <input type="checkbox"/> Portable GUI Development      |
| <input type="checkbox"/> Bayesian Statistics            | <input type="checkbox"/> Printer Technology Guide      |
| <input type="checkbox"/> Beowulf Clusters               | <input type="checkbox"/> Real Time Systems             |
| <input type="checkbox"/> C++                            | <input type="checkbox"/> Regression Testing            |
| <input type="checkbox"/> Client-Server Technology       | <input type="checkbox"/> Security and the Internet     |
| <input type="checkbox"/> COM                            | <input type="checkbox"/> Simulation                    |
| <input type="checkbox"/> Computational Fluid Dynamics   | <input type="checkbox"/> Soft Computing                |
| <input type="checkbox"/> Computer Image Processing      | <input type="checkbox"/> Software Design Methodologies |
| <input type="checkbox"/> Decision Support Systems       | <input type="checkbox"/> Software Development Cycle    |
| <input type="checkbox"/> Electronic Data Capture        | <input type="checkbox"/> Software Documentation        |
| <input type="checkbox"/> Electronic Lab Notebooks       | <input type="checkbox"/> Software Portability          |
| <input type="checkbox"/> Excel                          | <input type="checkbox"/> Software Re-engineering       |
| <input type="checkbox"/> Extending the Life of Software | <input type="checkbox"/> Software Specification        |
| <input type="checkbox"/> Federal Drug Administration    | <input type="checkbox"/> SQL                           |
| <input type="checkbox"/> FORTRAN 90                     | <input type="checkbox"/> UNIX Inter-Process Comms      |
| <input type="checkbox"/> Grid Computing                 | <input type="checkbox"/> UNIX Systems Performance      |
| <input type="checkbox"/> High Throughput Screening      | <input type="checkbox"/> UNIX Workstations             |
| <input type="checkbox"/> Instrumentation                | <input type="checkbox"/> Visual Basic 6                |
| <input type="checkbox"/> Integrated Lab Systems         | <input type="checkbox"/> WAP                           |
| <input type="checkbox"/> J2EE                           | <input type="checkbox"/> Web Services                  |
| <input type="checkbox"/> Java                           | <input type="checkbox"/> Windows 2000 Services         |
| <input type="checkbox"/> Lims                           | <input type="checkbox"/> XML                           |
| <input type="checkbox"/> Linux                          | <input type="checkbox"/> X Windows                     |
| <input type="checkbox"/> Microsoft Net                  |  |



INVESTOR IN PEOPLE

**Tessella Support Services plc**

3 Vineyard Chambers, Abingdon, Oxon, OX14 3PX, England

Tel: (+44) (0) 1235 555511 Fax: (+44) (0) 1235 553301

E-mail: [info@tessella.com](mailto:info@tessella.com) Web Address: <http://www.tessella.com>