Preface

In recent years, the cost required to develop a software product has increased significantly while the target hardware costs have decreased. Now a larger portion of money is expended in developing, using, and maintaining software. The trend in computing costs is the complete dominance of software over hardware costs. Because of this, it is necessary that formal disciplines be established to increase the probability that software is characterized by a high degree of correctness, maintainability, and portability. In addition, these disciplines must promote practices that aid in the consistent and orderly development of a software system within schedule and budgetary constraints. To be effective, these disciplines must adopt standards which channel individual software efforts toward a common goal.

The push for standards in the software development field has been met with various degrees of success. The Microprocessor Operating Systems Interfaces (**MOSI**) effort has experienced only limited success. As popular as the **UNIX** operating system has grown, the attempt to develop a standard interface definition to allow portable application development has only recently begun to produce the results needed in this area. Unfortunately, very little effort has been expended to provide standards addressing the needs of the real-time community. Several organizations have addressed this need during recent years.

The Real Time Executive Interface Definition (**RTEID**) was developed by Motorola with technical input from Software Components Group. **RTEID** was adopted by the VMEbus International Trade Association (**VITA**) as a baseline draft for their proposed standard multiprocessor, real-time executive interface, Open Real-Time Kernel Interface Definition (**ORKID**). These two groups are currently working together with the IEEE P1003.4 committee to insure that the

functionality of their proposed standards is adopted as the real-time extensions to **POSIX**.

This emerging standard defines an interface for the development of real-time software to ease the writing of real-time application programs that are directly portable across multiple real-time executive implementations. This interface includes both the source code interfaces and run-time behavior as seen by a real-time application. It does not include the details of how a kernel implements these functions. The standard's goal is to serve as a complete definition of external interfaces so that application code that conforms to these interfaces will execute properly in all real-time executive environments. With the use of a standards compliant executive, routines that acquire memory blocks, create and manage message queues, establish and use semaphores, and send and receive signals need not be redeveloped for a different real-time environment as long as the new environment is compliant with the standard. Software developers need only concentrate on the hardware dependencies of the real-time system. Furthermore, most hardware dependencies for real-time applications can be localized to the device drivers.

A compliant executive provides simple and flexible real-time multiprocessing. It easily lends itself to both tightly-coupled and loosely-coupled configurations (depending on the system hardware configuration). Objects such as tasks, queues, events, signals, semaphores, and memory blocks can be designated as global objects and accessed by any task regardless of which processor the object and the accessing task reside.

The acceptance of a standard for real-time executives will produce the same advantages enjoyed from the push for **UNIX** standardization by AT&T's System V Interface Definition and **IEEE**'s **POSIX** efforts. A compliant multiprocessing executive will allow close coupling between **UNIX** systems and real-time executives to provide the many benefits of the **UNIX** development environment to be applied to real-time software development. Together they provide the necessary laboratory environment to implement real-time, distributed, embedded systems using a wide variety of computer architectures.

A study was completed in 1988, within the Research, Development, and Engineering Center, U.S. Army Missile Command, which compared the various aspects of the Ada programming language as they related to the application of Ada code in distributed and/or multiple processing systems. Several critical conclusions were derived from the study. These conclusions have a major impact on the way the Army develops application software for embedded applications. These impacts apply to both in-house software development and contractor developed software.

A conclusion of the analysis, which has been previously recognized by other agencies attempting to utilize Ada in a distributed or multiprocessing environment, is that the Ada programming language does not adequately support multiprocessing. Ada does provide a mechanism for multi-tasking, however, this capability exists only for a single processor system. The language also does not have inherent capabilities to access global named variables, flags or program code. These critical features are essential in order for data to be shared between processors. However, these drawbacks do have **workarounds** which are sometimes awkward and defeat the intent of software maintainability and portability goals.

Another conclusion drawn from the analysis, was that the run time executives being delivered with the Ada compilers were too slow and inefficient to be used in modern missile systems. A run time executive is the core part of the run time system code, or operating system code, that controls task scheduling, input/output management and memory management. Traditionally, whenever efficient executive (also known as kernel) code was required by the application, the user developed in-house software. This software was usually written in assembly language for optimization.

Because of this shortcoming in the Ada programming language, software developers in research and development and contractors for project managed systems, are mandated by technology to purchase and utilize off-the-shelf third party kernel code. The contractor, and eventually the Government, must pay a licensing fee for every copy of the kernel code used in an embedded system.

The main drawback to this development environment is that the Government does not own, nor has the right to modify code contained within the kernel. V&V techniques in this situation are more difficult than if the complete source code were available. Responsibility for system failures due to faulty software is yet another area to be resolved under this environment.

The Guidance and Control Directorate began a software development effort to address these problems. A project to develop an experimental run time kernel was begun that will eliminate the major drawbacks of the Ada programming language mentioned above. The Real Time Executive for Military Systems (**RTEMS**) provides full capabilities for management of tasks, interrupts, time, and multiple processors in addition to those features typical of generic operating systems. The code is Government owned, so no licensing fees are necessary. **RTEMS** has been implemented in both the **Ada** and **C** programming languages. It has been ported to the Motorola MC68xxx family, the Intel i80386 and above, and the Intel i80960 family. Support for other processor families, including RISC, CISC, and DSP, is planned. Since almost all of **RTEMS** is written in a high level language, ports to additional processor families require minimal effort.

RTEMS multiprocessor support is capable of handling either homogeneous or heterogeneous systems. The kernel automatically compensates for architectural differences (byte swapping, etc.) between processors. This allows a much easier transition from one processor family to another without a major system redesign.

Since the proposed standards are still in draft form, **RTEMS** cannot and does not claim compliance. However, the status of the standard is being carefully monitored to guarantee that **RTEMS** provides the functionality specified in the standard. Once approved, **RTEMS** will be made compliant.

This document is a detailed users guide for a functionally compliant real-time multiprocessor executive. It describes the user interface and run-time behavior of Release 3.1.x of the C implementation of **RTEMS**.