KAPL-P-000204
(K97158)
CONF-980448--

REAL TIME PROGRAMMING ENVIRONMENT FOR WINDOWS

D. R. LaBelle

April 1998

NOTICE

KAPL ATOMIC POWER LABORATORY SCHENECTADY, NEW YORK 15801

Operated for the U. S. Department of Energy
by KAPL, Inc. a Lockheed Martin company

DISCLAIMER

REAL TIME PROGRAMMING ENVIRONMENT FOR WINDOWS

Dennis R. LaBelle
22 Sandalwood Drive
Clifton Park, NY 12065
e-mail drlabell@albany.net

KEYWORDS

real time simulation tool WindowsNT

ABSTRACT

This document provides a description of the Real Time Programming Environment (RTProE). RTProE tools allow a programmer to create soft real time projects under general, multi-purpose operating systems. The basic features necessary for real time applications are provided by RTProE, leaving the programmer free to concentrate efforts on his specific project. The current version supports Microsoft Windows™ 95 and NT. The tasks of real time synchronization and communication with other programs are handled by RTProE. RTProE includes a generic method for connecting a graphical user interface (GUI) to allow real time control and interaction with the programmer's product. Topics covered in this paper include real time performance issues, portability, details of shared memory management, code scheduling, application control, Operating System specific concerns and the use of Computer Aided Software Engineering (CASE) tools.

The development of RTProE is an important step in the expansion of the real time programming community. The financial costs associated with using the system are minimal. All source code for RTProE has been made publicly available. Any person with access to a personal computer, Windows 95 or NT, and C or FORTRAN compilers can quickly enter the world of real time modeling and simulation.

GOALS

Having done real time application development under a couple of versions of UNIX, I set out to investigate what the 32-bit versions of Windows could do. RTProE is the product of this investigation.

The project goals consisted of:

1. Creating an automated system for programming real time applications such as simulators and non-critical, medium speed control systems.
2. Maximizing multi-platform portability
3. Producing freely available source code
4. Generating efficient, compact, low maintenance source code
5. Providing a well integrated Graphical User Interface for real time applications.

PERFORMANCE ISSUES

Overview

The Win32 environment has most of the important real time features found in modern versions of UNIX. These include:

1. Shared memory
2. Timer interrupts
3. Priority modification
4. Program memory locking
5. Assignment of a program to a specific CPU

The first 3 items are by far the most important and are currently the only ones used by RTProE. Efforts to lock the program in memory are generally unnecessary in today's high RAM environments. CPU assignment is only beneficial on multi-processor systems and would be easily implemented with a single function call.

Shared Memory

Shared memory is by far the preferred method of inter-process communication for this real time project. The use of shared memory makes for an extremely simple and portable design. Only minor changes are necessary to adapt the Windows shared memory usage to UNIX.

Shared memory uses little CPU overhead and is extremely fast. However, **most importantly**, it allows inter-process communication without using any of the Windows Application Programming Interface (API) function calls. This avoids Windows' annoying tendency to *steal* the processor away from an application in order to service low priority events such as mouse movements. An application's use of any other type of inter-process communication under Windows severely degrades its real time response. However, avoiding the use of the Windows API functions produces a surprisingly robust real time application.

Timer Interrupts

Windows 95 and NT are capable of providing timer interrupts at a resolution of 1 millisecond. From informal statements noted by other users on the Internet, the interrupt precision is normally within 1 millisecond and sometimes as bad as 10 milliseconds.

Priority Modification

RTProE uses the maximum level of the Windows REALTIME_PRIORITY_CLASS to ensure proper response of the real time application. This, of course, could lead to a dangerous situation should the application programmer write faulty code with an infinite loop or infinite wait condition. RTProE includes special code to detect such conditions and automatically breaks out of them. This RTProE feature works quite well even with a loop such as:

```
while (1);
```

Windows 95 versus NT

Running real time application code under both Windows 95 and NT demonstrates a significant difference in real time response between Windows 95 and NT. Windows NT is much better at honoring the requested REAL TIME priority level and responding to the timer interrupt than Windows 95. Windows 95 performance appears satisfactory if CPU loading by the real time application is kept under 40 percent.

PORTABILITY

Producing a multi-platform compatible product is one of the major goals of RTProE. Subsequently, many of its utilities are command line based. Graphical user interfaces were constructed using the TCL/TK programming language. TCL/TK code can be run unmodified under Windows, UNIX and the Mac operating system. Therefore, TCL/TK represents an excellent choice for a multi-platform product. The data base system, build environment and the Interactive Control Station (ICS) portion of RTProE, have graphical interfaces.

The first version of RTProE was produced under Windows 95/NT. Where more than one method of implementation was possible under Windows, the choice was made in favor of the approach most compatible with the UNIX operating system. This was done to simplify the migration path to UNIX.

SOURCE CODE

Except for graphical user interface portions, done in TCL/TK, RTProE has been written in C. The specific compiler used was Microsoft Visual C++ version 4.0. To maintain portability to UNIX, there was no use made of Microsoft Foundation Class libraries.

Much of the code was created using PC versions of the LEX and YACC utilities. Public domain versions of these long, time UNIX tools were used to generate the code for the programs *hgen, igen, pdm and rtdata*. This resulted in a relatively small amount of code that must be maintained for RTProE (~6000 lines). This code takes the form of input files to the LEX and YACC programs.

The choice of LEX and YACC was a natural one. The bulk of the work in creating a real time programming environment consists of routine computer science lexical translation and grammar parsing. These are tasks LEX and YACC were respectively designed for.

All source code for RTProE is included in Appendix A of this document.

PROGRAMMING LANGUAGES SUPPORTED

RTProE currently supports the C and FORTRAN programming language for real time user applications.

RTProE MEMORY MANAGEMENT

Overview

The *pdm* program manages a data base of variables which will be used by a real time application. This data base is known as the Programmer's Data Base (PDB). The variables specified in the PDB will exist in shared memory when the application is run. Using RTProE's function library and tools, other applications can access and interact with these variables during run time. Since RTProE uses shared memory exclusively for communication with the real time application, access to this data is provided with no run time speed penalty.

The PDB contains the following information about program variables:

1. Variable name
2. Data type
3. Array dimensioning
4. Name of shared memory structure in which variable resides
5. Location in shared memory structure

The *hgen* and *igen* programs generate header files of variable declarations for each file that uses variables maintained in the PDB. Variables contained in the PDB must not be declared in the source code by any other method.

The header generation programs use information in the PDB to produce data structures which contain the PDB variables found within a source code file. These structures are called global partitions. The header files produced contain the necessary compiler directives to place the global partitions in shared memory. The program source code should use the PDB variables as discrete, non-structure items. The necessary structure syntax will be handled by statements in the, automatically generated, header file.

Global Partitions

RTProE manages the placement of user application variables into shared memory. With shared memory, the information in the variables can be made available to any other running program. The information directly manipulated by the application program, resides in shared memory. The shared memory variables are not copies. They are the actual data items used by the application program. Therefore, the use of shared memory results in no additional processing overhead for the communication of the data between programs. This provides the fastest possible data transfer available between programs.

Large data structures are used for the efficient handling of this shared memory. The actual location of a specific variable within the shared memory structures is determined by the PDB. The structures are called global partitions and are automatically created using the *hgen* and *igen* programs described later.

Partition Types - There are three general types of global partitions. A programmer uses *pdm* to assign a variable to a partition type based on its functionality. The three global partition types and the variables they should contain are described below:

Global Partition for Non-save Variables - These variables are placed in shared memory but do not define the "state" of the application. They are modified as part of the program execution but only reflect a temporary condition of the application. The variables have global scope within the application. This type of partition is never saved to or restored from a file.

Global Partition for Initial Condition Variables - These variables are placed in shared memory. Their values are necessary for restoring the "state" of the application. They are modified as part of the program execution. The variables have global scope within the application. All partitions of this type are saved to file when an Initial Condition write request is made. This write request can be made using the *rtdata* command SNAP. All partitions of this type are restored from file when an Initial Condition read request is made. This read request can be made using the *rtdata* command RESET.

Global Partition for Constant Coefficients - These variables are placed in shared memory. They define the behavioral characteristics of the application and are not modified by execution of the program. However, their values may be changed by the programmer to modify the behavior of the program. The variables have global scope within the application. All partitions of this type are recalled from file when the real time application is first run. All partitions of this type are copied from memory to the file *globcon.dat* by using the *rtdata* command CONSAVE. The RTProE real time scheduler (rtsched.c) looks for this data at the end of the user application executable file. The last part of building the real time user application involves attaching *globcon.dat* to the end of the executable file. This is one of the tasks performed by the build program or the RTProE Workbench.

Either the ICS or the RTDATA program can be used to set the values of the constants before creating the globcon.dat file.

Defining a Partition - Each global partition must have an entry in the PDB. The information for a partition is entered like any other program variable, using *pdm*. The user may define any number of each global

partition type. Global partitions are normally defined as large, single dimension arrays. For each item of the array, 32 bytes of shared memory will be reserved by the user application at run time. These shared memory structures should be created large enough to hold all the variables the user wishes to place in them. However, *pdm* allows the user to modify the size of the partition with its MOD command.

Reserving Space for a Variable Within a Partition - Space is reserved within a partition by creating an entry in the PDB for the variable. When first defining a variable, the user must also assign it to a global partition. Placing the variable in the PDB with the *pdm* reserves space in the shared memory data structure for the variable.

PDB addition and modification may be made interactively or in a batch mode. This last method provides for the import of data from different RTProE data bases or other real time development systems.

Shared Memory Under Windows95/NT - In order to create and use shared memory under Windows 95 or NT, RTProE performs the following:

1. Uses *hgen* to create header files for C program source code. Uses *igen* for FORTRAN source code.

   These header files contain the global partition structures to properly access the parameters in shared memory.

2. Uses *hgen* to create the special header file for rtsched.c.

   This special header contains the necessary compiler directives for global partition initialization and export to other programs.

3. Uses the RTProE Windows resource file *rtsched.rc*

   This file defines the global partition structures as shared memory.

4. Compiles the real time application

After building the application, the user should place the *rtdata.exe* and *rtsched.exe* files in a PATH directory prior to use. The *rtdata* program needs to access the real time application file at startup in order to access its shared memory. Placing *rtsched.exe* in one of the PATH directories is one method of ensuring it can do so.

SUPPORTING PROGRAMS

RTProE consists of several supporting utilities. Most of these programs are automatically controlled and executed through the RTProE Workbench program. Summary descriptions of the utilities follow.

Build

The TCL/TK language was used to produce a program that can automatically perform the proper scanning of source code and building of a real time program with RTProE. The program is called *build*. Although Make files are traditionally used to maintain and build applications, the capabilities of the Make utility varies between operating systems and vendors. Therefore, the cross-platform scripting language TCL/TK proved an excellent choice for producing a portable utility.

Build performs source/target comparisons and compilations similar to Make. The program also manages a hierarchy based on "Official Development" and "User Development" areas. The bulk of a real time application is normally assembled from an "Official Development" area with small modifications coming from a "User Development" subdirectory.

Framegen

Overview - The *framegen* program generates the necessary C source code for scheduling the programmer's code at intervals specified in the file *frameseq.in*. The frame sequencing input file has a simple text file format. The generated source code is stored in the file *frameseq.c*. The frame code generator can produce source code for any number of frames per second.

RTProE supports the execution of user code fragments at multiple frequencies. The *frameseq.in* file is used to identify the desired frequency for each portion of the real time application.

Scheduling Real time Applications - Real time application programs normally divide execution time into an integral number of frames per second. During each frame, the application performs some work. After the work is completed, the real time application suspends itself until the start of the next time frame. Processing the *frameseq.in* file allows the programmer to specify:

1. The total number of frames per second.

   This is the number of time slices which the programmer wishes to schedule during each second. The minimum number is 1. Under Windows 95/NT the maximum number is 1000. The frames per second is specified when processing the *frameseq.in* file with *framegen*. The *framegen* program can accept a single argument on the command line. This argument is the number of time slices (frames) that will be scheduled during each second.

2. How many times per second a subroutine is called.

   Individual subroutines do not need to be scheduled for execution every time frame. To conserve CPU, they should only be scheduled as often as the programmer deems necessary. For example, the program may need to update a displayed numeric value only once per second.

3. Load balancing on a per second basis.

   The work performed by a real time application is normally spread evenly across time. This presents the best real time behavior to the user or other interfacing applications. The programmer should not schedule all the subroutine calls into a single frame.

   Load balancing is performed through the use of execution groups. An execution group is an evenly spaced subset of the total frames available per second. The number of groups is dependent on the total number of frames in a second. For example, if 12 frames per second were specified, there would be:

   12 groups with an execution rate of 1 frame per second
   6 groups with an execution rate of 2 frames per second
   4 groups with an execution rate of 3 frames per second
   3 groups with an execution rate of 4 frames per second
   2 groups with an execution rate of 6 frames per second
   1 group with an execution rate of 12 frames per second

   The table below shows the group numbers and execution frame number for each calling frequency available in a 12 Hertz program.

Execution Frame =>	1	2	3	4	5	6	7	8	9	10	11	12
12 Hertz	1	1	1	1	1	1	1	1	1	1	1	1
6 Hertz	1	2	1	2	1	2	1	2	1	2	1	2
4 Hertz	1	2	3	1	2	3	1	2	3	1	2	3
3 Hertz	1	2	3	4	1	2	3	4	1	2	3	4
2 Hertz	1	2	3	4	5	6	1	2	3	4	5	6
1 Hertz	1	2	3	4	5	6	7	8	9	10	11	12

As seen in the table above, if a subroutine was scheduled for execution 4 times per second in group 2, it would run during frames 2, 5, 8 and 11.

In scheduling a subroutine for execution 4 times per second, a decision must be made as to which of 3 groups to use. Do not schedule all 4 Hertz activity to the same group. Doing so would create a load imbalance. Instead, it is better to alternate the groups to which the 4 Hertz subroutine calls are assigned.

The *framegen* program reads *frameseq.in* and produces the C source code to correctly schedule the user's subroutines as requested. This generated code is stored as file *frameseq.c*.

Hgen

The *hgen* program scans the programmer's C source code and generates the necessary header file for compilation into the real time application. Any variable names which appear in both the user's source code and the PDB will be identified and the necessary header file information will be produced.

Generation of include files for FORTRAN code is done by the program *igen*.

The *hgen* program reads its data from standard input (stdin) and generates the header file information to standard output (stdout). Therefore, the redirection symbols are used to provide input and indicate the desired location of the output. The output is redirected to a header file associated with the source code file being scanned. The scanned source code must also reference the header file created with *hgen*.

Example - The following example shows how *hgen* is used to scan the file *xxc01a.c*. The program variables xxtemp1 and xtemp2 have already been defined in the PDB.

Input source code file: xxc01a.c

```
#include "xxc01a.h"

void xxc01a()
{/*This subroutine increments a few values once per second */
 ++xxtemp1;
 xxtemp2 = xxtemp1 % 12;
}
```

Command Line to Generate Header File

```
hgen <xxc01a.c >xxc01a.h
```

Resulting Output Sent to File: xxc01a.h

```
#pragma data_seg("SHAREDAT1")

__declspec (dllexport) struct {
                    long int xxtemp1;
                    long int xxtemp2;
                    char dummy0[32760];
                   } global01;

#pragma data_seg()

#define xxtemp1 global01.xxtemp1
#define xxtemp2 global01.xxtemp2
```

Ics client

The Interactive Control Station client, *ics*, provides a graphical interface for interactive control of real time applications produced with RTProE. The *ics* program communicates with the Interactive Control Station server, *icserver*, to obtain information and control the real time program. This is done through TCP/IP sockets, allowing easy communication over a network.

Ics is written in TCL/TK. TCL 7.6 and TK 4.2 or later revisions must be installed on the system to run it.

The author discusses the *ics, icserver* and *icsgraph* programs in greater detail in the accompanying paper "*Building a Simulator Control Station using the TCL/TK Language*".

Icserver

The interactive control server (icserver) provides a set of centralized services for controlling the real time application locally or over a network. It uses the *rtdata* program for communication with the real time program. *Icserver* is written in TCL/TK and communicates with clients using TCP/IP sockets.

Icsgraph

The *icsgraph* program allows the user to plot the values of variables contained within the real time application. This plotting is performed in real time, while the application runs. The variables must be defined using the *pdm* program. Any of the variables defined in the *pdm* data base can then be specified by their text label during run time.

The following features are available with *icsgraph*:

1. Selection of up to 4 plotted parameters from the PDB
2. Automatic scaling and run rate adjustment
3. Sample rate adjustment
4. Display range selection
5. File SAVE and LOAD of parameter sets

Multiple copies of *icsgraph* may be run simultaneously. *Icsgraph* is written in TCL/TK and communicates with the *icserver* using TCP/IP sockets.

Igen

The *igen* program scans the programmer's FORTRAN source code and generates the necessary include file for compilation into the real time application. Any variable names which appear in both the user's source code and the PDB will be identified and the necessary include file information will be generated.

Generation of header files for C code is done by the program *hgen*.

The *igen* program reads its data from standard input (stdin) and generates the header file information to standard output (stdout). Therefore, the redirection symbols are used to provide input and indicate the desired location of the output. The output is redirected to an include file associated with the

source code file being scanned. The scanned source code must also reference the include file created with *igen*.

Example - The following example shows how *igen* is used to scan the file xxc01b.f. The program variables i_4byte, i_4byte_2 and i_4byte_3 have already been defined in the PDB.

Input Source Code File: xxc01b.f

```
      subroutine xxc01b
C This subroutine increments a few values once per second.
      include "xxc01b.inc"

      i_4byte   = i_4byte + 1
      i_4byte_2 = i_4byte + 1
      i_4byte_3 = i_4byte_2 + 1
      END
```

Command Line to Generate Include File

```
igen <xxc01b.f >xxc01b.inc
```

Resulting Output Sent to File: xxc01b.inc

```
      character*1 GLOBAL03X(64)
      COMMON /GLOBAL03/GLOBAL03X

      integer*4 i_4byte
      EQUIVALENCE(i_4byte, global03X(5))

      character*1 GLOBAL00X(2048)
      COMMON /GLOBAL00/GLOBAL00X

      integer*4 i_4byte_2
      EQUIVALENCE(i_4byte_2, global00X(1))

      integer*4 i_4byte_3
      EQUIVALENCE(i_4byte_3, global00X(5))
```

Pdm

The *pdm* program manages a data base of variables which will be used by a real time user application. This data base is known as the Programmer's Data Base (PDB). The variables specified in the PDB will exist in shared memory when the application is run. Using RTProE's function library and tools, other applications can access and interact with these variables during run time. Access to this data is provided with no run time speed penalty. The *pdm* program is generated using LEX and YACC.

The *pdm* is used to manage the placement of application program variables within large shared memory data structures. These structures are called global partitions. The *pdm* provides the following functionality for each variable defined within it:

1. Identification of a variable as either:
   a. A temporary variable
   b. A state variable
   c. A constant coefficient
2. Automatic placement of the variable into an available location within a partition
3. Automatic generation of the variable's data definition within a header file
4. Maintenance of programming specific information
   a. Variable name
   b. Data type
   c. Array dimensioning
   d. Name of shared memory partition
   e. Offset into shared memory partition
5. Numeric display formatting
6. Storage of descriptive, non-programming data

Under Windows, the *pdm* program is written as a 32-bit console application with a keyboard interface. Prior to starting *pdm*, the environment variable ODSPATH must be defined. This is the only environment variable required for using RTProE. ODSPATH must specify the directory in which the data base files are maintained. Multiple projects may be maintained on the same system by storing the data base files in separate directories. The value of ODSPATH can then be changed, as necessary, to switch to another project.

Pdm is started in the same manner as any other Windows program. When run interactively, the user is presented with a console window. *Pdm* does not provide a command line prompt. However, it will accept certain commands and provide some feedback. Although it may be accessed directly through its command line interface, the graphical front end provided by the RTProE Workbench is normally used.

Rtdata

The *rtdata* program acts as a data communications path between the real time application and other programs. It is used to control the application run state and read or modify values in the real time program. Using RTProE's function library *rtdata* can access and interact with the variables in the real time application.. Access to this data is provided with no run time speed penalty. The *rtdata* program is generated using LEX and YACC.

The *rtdata* program uses the Programmer's Data Base (PDB) to determine the placement of variables within shared memory data structures used by the real time application. These structures are called global partitions. The user, however, refers to the data location by its normal symbolic name in the application.

Under Windows, the *rtdata* program is written as a 32-bit console application with a keyboard interface. *Rtdata* is started in the same manner as any other Windows program. By default, *rtdata* will connect to the shared memory of the program *rtsched.exe*. To connect to the shared memory of some other RTProE real time application, specify the program name on the *rtdata* command line.

Example: **rtdata myapp.exe**

When run interactively, the user is presented with a console window. *Rtdata* does not provide a command line prompt. However, it will accept certain commands and provide some feedback.

Rtdata is not normally accessed directly through the console. The graphical user interface (GUI) program, Interactive Control Station (ICS), should be used to access *rtdata*'s features. ICS runs *rtdata* as a child process and communicates with it through stdin and stdout.

List management - The main purpose of *rtdata* is retrieval of the current value of program variables from shared memory. A list management scheme is employed for efficient retrieval of this information using *rtdata*. The user first defines a list of program variables to retrieve on a regular basis. All values of items in the list can then be obtained with a single, short command (GETLIST). Multiple lists may be maintained simultaneously.

There are three commands (NEWLIST, ADDTOLIST AND GETLIST) devoted to managing lists of variables needing retrieval. The NEWLIST command creates a new, empty list and returns its name in the form of a number. This number is referred to as the list_id and must be used with the other two list commands. The list_id specifies which list a command should access.

For infrequent, non-list based, data retrieval the GETVAL command is available.

Rtsched.c

The basic real time scheduling control is obtained by using the main() function of *rtsched.c*. Real time programs are generated by using *rtsched.c* for the main control loop. This file contains the code for all of the real time scheduling and control features of RTProE. The main features include:

1. Freeze (suspend) the application
2. Run (resume) the application
3. Save the current state of the application
4. Reset (restore) the application to a previous state
5. Terminate the application
6. Multi-frequency function calling rates

Subroutines

main()

Under Windows, *rtsched.c* is implemented as a Win32 console application. It, therefore, has main() as the program entry point. The main() function performs the following tasks:

1. Change process to REALTIME_PRIORITY_CLASS
2. Set up periodic frame timer
3. Create Event for controlling real time loop
4. Create real time control thread
5. Use Sleep function to relinquish all control to real time thread

FrameMsg()

Windows' timeSetEvent function is used to specify a callback function which will be executed at a specific frequency. The time interval between calls to FrameMsg() is the referred to as a frame.

The callback function, FrameMsg(), activates the real time control thread using Windows' PulseEvent() function. Therefore, the control thread is activated once per frame.

FrameMsg() also determines whether the application is successfully running in real time. It determines whether the real time activity is completing within the frame. Failure to complete the real time activity within the assigned frame is considered an overrun. FrameMsg() allows the real time control loop to catch up using spare CPU capacity. However, if FrameMsg() determines the real time loop is lagging real time by an excessive amount, it will terminate the real time control thread, FREEZE the real time application and start a new real time control thread. At this point, the state of the real time application is still fully available for review.

This overrun detection capability is particularly useful for detecting and recovering from infinite loops or infinite wait conditions in the user's application. Without this feature, it would be especially difficult to regain control of the computer since the real time application is running at the highest priority level.

rtctrl()

This subroutine controls the performance of the following actions based on the value of the STATUS and ICRW variables:

1. Freeze (suspend) the application
2. Run (resume) the user application code
3. Save the current state of the application
4. Reset (restore) the application to a previous state
5. Terminate the application

ICsave()

This subroutine will save, to file, all variables which have been identified in the Programmer's Data Base as defining the state of the application. Rtctrl() will call this subroutine when the ICRW variable is set to a value of 'W'. However, the name of the save file must first be placed in the character array ICNAME.

ICload()

This subroutine will restore, from file, Initial Condition (IC) variables which have been saved with the *ICSave()* subroutine. *Rtctrl()* will call this subroutine when the ICRW variable is set to a value of 'R'. However, the name of the file to retrieve must first be placed in the character array ICNAME.

load_constants()

The *main()* function calls *load_constants()* at start up to retrieve values for the user application constant coefficients. These values are not meant to change during execution of the user application. However, it may be desirable to alter the coefficients during development of the application in order to produce the desired mathematical result. The coefficients are variables which are read but not modified by the application. The constant coefficients are modified by the programmer using RTProE tools. This may be done either at application run time or off-line. These special case, constant variables are identified using the *pdm* program.

CONCLUSIONS

The 5 main goals of the project were reasonably achieved. It is now possible to quickly write a compact, full featured, low cost real time development environment with graphical user interface. This is due to the availability of many free development tools along with recent hardware and operating system advances.

The development of RTProE is an important step in the expansion of the real time programming community. The financial costs associated with using the system are minimal. Any person with access to a personal computer, Windows 95 or NT, and C or FORTRAN compilers can quickly enter the world of real time modeling and simulation.

ABOUT THE AUTHOR

Dennis LaBelle is currently employed as a software engineer for Lockheed Martin. He holds a B.S. in Chemical Engineering from the University of Maine and received his M.S. in Computer Science from Rensselaer Polytechnic Institute. The author has 16 years programming experience in a wide variety of mainframe, Workstation and personal computer environments.