

# Embedded Systems Programming Must it be so tough?

*Embedded systems programming typically requires dedicated development tools, that often run as command line programs only. The INtime real time extension to Windows NT provided by RadiSys Corporation not only adds determinism to Windows NT, but also includes development support similar to what the Windows NT programmer takes for granted these days. This article provides details of the INtime development tools, and describes some of the implementation methods.*

## IN THE BEGINNING...

**M**any years ago, when I started programming, we had pre-printed forms of 25 lines and 80 columns and every programmer would carefully spell out the assembly language statements and comments on these forms. The forms were then taken to the data-entry department and after some days you received a deck of punched cards, which you had to verify and feed to the computer. To correct a program you had to punch small quantities of cards yourself; this protracted process of editing and compiling went on for many cycles. With this style of coding, I would think carefully before having the actual code punched. Some years later interactive programming entered the field via BASIC interpreters. Immediately we adjusted our style and now we did not think at all but started entering this sequence:

```
100 BEGIN
1000 END
      RUN
```

and if that did not fully implement the program requirements, we started patching. Sometimes we got such a program working, but the result was usually a poorly structured program.

Today developers of PC software expect interactive development tools with fancy features such as syntax colouring, context-sensitive help, etc. We even have program skeleton generators like the application wizards in Microsoft's Developer Studio. But because many operating systems outside the PC world have their own development tools and executable file formats, many developers of embedded system

code have to rely on command line compilers and similar tools to do their coding.

In this article, I will describe how the INtime product, a real-time extension to Windows NT by RadiSys Corporation, addresses the embedded application development issue and what steps we have taken to make the life of the embedded programmer easier.

INtime: Real Time for Windows NT

Just in case you have not met the INtime product yet, let me introduce it: Windows NT by itself lacks determinism (sometimes called hard real-time capabilities) amongst others because of the way it deals with hardware interrupts (for details on this, see the web site [www.realtime-info.com](http://www.realtime-info.com)). By adding INtime software to a Windows NT system, applications can be developed and executed that live partly in the well-known Win32 environment, but on the other hand have access to lower level system elements such as I/O ports and interrupts. INtime software combines the accepted

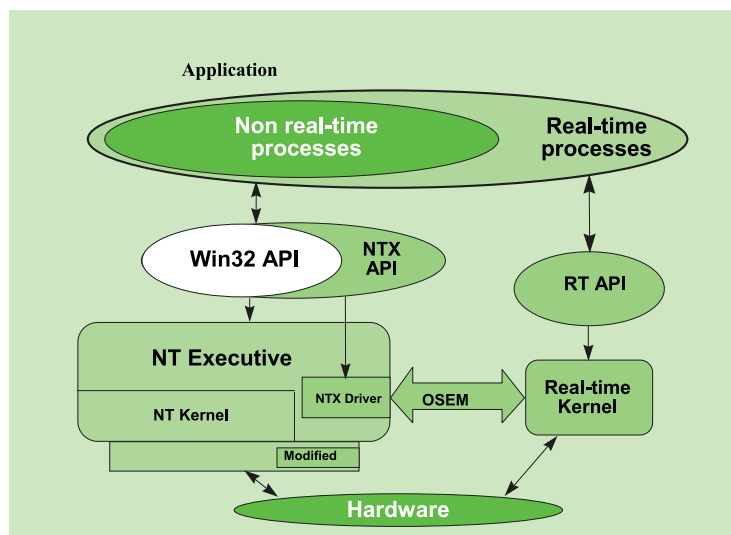


Figure 1. INtime software and Windows NT

graphical user interface that all Windows users know and expect with the deterministic features of a hard real time operating system.

The INtime implementation uses a mechanism called OSEM (Operating System Encapsulation Mechanism, see figure 1) to take over control of the CPU, while it still appears to Windows NT as if it controls all hardware. OSEM encapsulates all of Windows NT as the lowest priority real time thread and hence offers enough opportunities for real time threads to be more important than Windows NT. OSEM is implemented as a Windows NT kernel driver and also includes some modifications to the Hardware Abstraction Layer or HAL. Windows NT applications can use an extended API (named NTX) to communicate with real time applications.

The iRMX kernel is at the heart of the INtime product. iRMX kernel technology has been proven reliable for nearly twenty years in thousands of projects and millions of embedded systems and is still being applied in new systems. But because our target audience may not be familiar with iRMX, we wanted to introduce the real-time application interface smoothly into the Windows NT programmer's world. After all, the combination Windows NT plus INtime software may frequently be used by a programmer better acquainted with the Windows NT paradigm than with programming for a real-time operating system.

A primary goal of the INtime product is to simplify real-time program development. The INtime approach consists of five elements:

1. Define a real-time API in the style of the Windows NT API (also known as Win32)
2. Leverage state-of-the-art tools in real-time development
3. Provide tools that enforce a robust real-time programming model
4. Provide context sensitive help for program development.
5. Simplify changes to the program structure

While designing these elements, we had to decide which programming languages and compiler(s) to support. The choice for a programming language was simplified by the current preference for C (and C++); although there are attempts to use Java in embedded systems, the resulting performance is not good enough yet. Since Microsoft's Developer Studio is by far the most popular C/C++ compiler for Windows NT programming and we are targeting Windows NT programmers, we decided to use that compiler for developing real time applications. Developer Studio knows its own evolution, so currently we support both versions 4.2 and 5.0 of Developer Studio. All information in this article refers to version 5.0, but all features (except add-ins) have also been implemented in a similar fashion for version 4.2 of the compiler.

## **AD EMBEDDED SYSTEMS GERMANY 1**



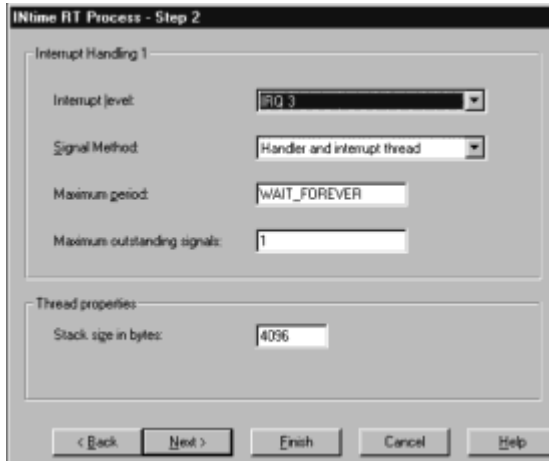


Figure 3. Sample of a wizard dialog

Developer Studio recognizes the DLL as a wizard and adds it to the list it displays when you create a new project (see figure 2). When you select a wizard from the list, Developer Studio activates it and passes a pointer to a COM object that allows the wizard to perform standard compiler functions.

A wizard enters in one or more dialogs where the programmer selects options (see figure 3); in some cases dialogs may be shown more than once. When the wizard has collected all program details, it enters the generation phase. The wizard sets macro variables and an interpreter generates a project with files and lines within these files; the wizard then terminates and Developer Studio regains control.

Although the wizard macro language is fairly simple, it provides enough features to generate all sorts of source lines and project content. The main interface is a project inventory (NEWPROJ.INF), which defines the constituents of the project - macro commands not only allow conditional inclusion of files, but they also permit customization of file names. Each file to be generated then requires a template and again using macro commands, this provides for generating application-specific source lines. Finally the COM interface allows the wizard to manipulate project settings (although unfortunately not all settings can be specified).

In particular the project settings feature is useful for generating real-time programs, as our different runtime environment requires some modified settings for the compiler and linker. Making these changes to the compiler and linker switches can be safely delegated to the wizard (in the first release of the INtime product we did not even document the settings and since nobody noticed that oversight, every developer must have used the wizard).

To illustrate this, here is a piece of our NEWPROJ.INF file:

```
$$BEGINLOOP (POLLS)
+POLL.C Poll$$P_NAME$.c
$$ENDLOOP
```

This sequence generates as many files as the macro variable POLLS holds and for each one uses the file POLL.C as a template. The result of processing the

template is written to a file with a name consisting of "Poll", the contents of the macro variable P\_NAME and the file extension .c; the resulting filename can be something like Poll1.c. This file is then added to the project (this addition is indicated by the "+" character before the template name). Inside the POLL.C template, a sequence like the following occurs:

```
$$IF (P_SLEEP)
    RtSleep ($$P_PERIOD$$);
$$ELIF (P_KERNEL)
    if
    (!knWaitForRtAlarmEvent (hAlarm$$P_NAME$$,
KN_WAIT_FOREVER))
        Fail ("Cannot wait for alarm
    $$P_NAME$$");
$$ENDIF
```

As a result one or the other code sequence is included in the output file, with macro variable substitution performed wherever necessary.

Currently the INtime product adds two wizards to Developer Studio (one for each style of real time application). For each wizard the programmer can choose between C and C++ as target programming language.

## THE INTIME PROGRAMMING MODEL

Every program, whether intended for Windows NT or for a real-time environment, should be written in a defensive style; e.g. it should clean up all its elements at program end and should recover from any initialization or cleanup errors. In the INtime product we recommend a certain programming style, which also provides a common method for "asking a program to terminate itself properly." A simplified code fragment illustrates this:

```
#include <rt.h>
RTHANDLE          hDelMbx;
void main(void)
{
    BYTE          byDeletionMsg[128];
    // create and catalog the deletion
    mailbox
    hDelMbx =
    CreateRtMailbox(DATA_MAILBOX | FIFO_QUEUE-
ING);
    if (hDelMbx == BAD_RTHANDLE)
        Fail("Cannot create deletion mail-
        box");
    if (!Catalog(NULL_RTHANDLE, hDelMbx,
    "R?EXIT_MBOX"))
        Fail("Cannot catalog deletion mail-
        box");
    // create threads and other objects
    here
    // wait for the kill message
    ReceiveRtData(hDelMbx, byDeletionMsg,
    WAIT_FOREVER);
    // cleanup the environment
    Cleanup(); // does not return
}
```

## DEV. ENVIRONMENT

Knowing programmers and their habits for producing "compact and efficient" code, the only way to get them to accept a given style, is to make it free. This is where wizards come in handy: they generate a program, which has all the elements of the model we recommend; and since these elements are concentrated in a part of the program where the programmer will not need to look very often, there is only a small chance that a programmer will disturb the model.

### DEVELOPER STUDIO ADD-INS

After INtime users express their appreciation for wizards, their next remark is usually that they would like an 'after-initial-setup-wizard': having used the wizard to create a program skeleton they want to be able to use something similar to add or modify elements of their program. With Developer Studio version 5.0, Microsoft introduced the concept of 'add-ins'. An add-in is a macro or DLL that extends the feature set of Developer Studio.

An add-in must first be installed and enabled (see figure 4); it can then be activated by a button just like many other Developer Studio functions. Whenever the

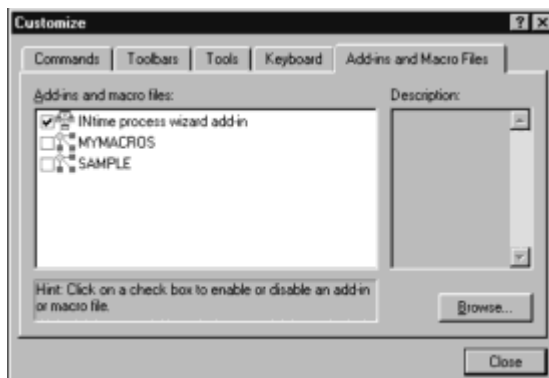


Figure 4: Setting up an add-in

programmer clicks such a button, Developer Studio performs sanity checks on the add-in and then activates it with a pointer to a COM object. That object allows the add-in to access many Developer Studio objects.

Since we supply both the wizards and the add-ins, we can pass information between them. We do this in a manner similar to the MFC application wizard style. Special source markers (which look like comments to the compiler) are inserted at critical locations and this allows the add-in to recollect the most critical information about the program structure. As an example, the following is what the wizard will generate when a polling thread is needed:

```
//{{POLLS : members
    DECLARE_POLL(1)
//}}
```

The add-in can easily browse the source (using the COM object it obtains at its activation) and can then

display a summary to the programmer (see figure 5), after which elements of the list can be removed or modified, or new elements can be added.

The add-in provided with the INtime product allows for modification of some parameters of existing elements,

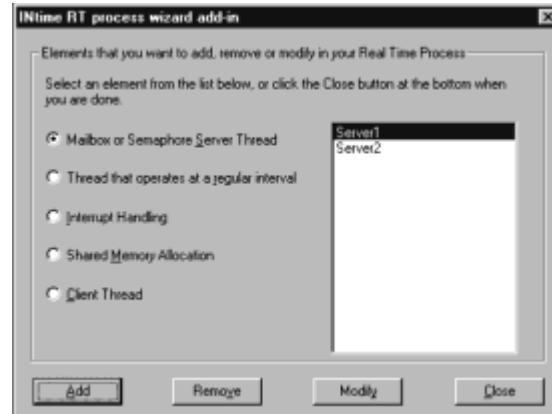


Figure 5: Sample of an add-in dialog

complete removal of an element, or adding a new element. When a file must be added or removed from a project, the current Developer Studio support does not allow the add-in to do this, so instead it displays a message and lets the user take over. Hopefully Microsoft will add the lacking functions here in a future release.

### CONCLUSION

For most programmers who have been confined to embedded software development, the development support described so far is a huge step forward. Seasoned Windows NT programmers will take these development tools for granted, but will therefore feel right at home when developing real-time Windows NT applications with the INtime product. This investment in providing familiar extensions that integrate directly with standard Windows NT development tools is already paying off handsomely in the form of brisk product acceptance and substantially easier development of real-time Windows NT applications. ■

---

*Jan Baan is specializing in software products for RadiSys Corporation in Europe and as such he is the technical contact for the INtime real-time extension for Windows NT. Besides evangelizing and supporting INtime, he also takes actively part in the further development of INtime and other software products.*

*Before joining RadiSys in 1996, he worked for many years with Intel Corporation in technical positions concentrating on operating systems such as iRMX (real-time), OS/2 and Windows NT; responsibilities were pre- and post-sales support, customer training and consulting. Earlier positions were in system software development at a Netherlands-based software house and at Philips Computer Industry.*