AN2949
Rev. 1, 1/2006

Using Processor Expert to Develop a Software Real-Time Clock

By: Steven Torres
MCU System/Application Engineering
Austin, Texas

Table of Contents

1 Introduction

This application note uses Processor Expert embedded beans and an HCS08 microcontroller (MCU) to demonstrate a software-based RTC (real-time clock). Processor Expert is a tool that can help reduce development time for embedded software creation and get products to market faster. This document provides a basic overview of Processor Expert and demonstrates its ease of use. A discussion of the M68DEMO908GB60 development tool is also provided.

2 Processor Expert and Embedded Beans

This section provides an overview of Processor Expert and embedded bean basics.

Processor Expert is a optional software plug-in for Freescale’s CodeWarrior development tools. Processor Expert provides object-oriented programming for embedded systems to facilitate rapid application development. With Processor Expert, MCU peripherals are configured through a graphical user interface (GUI) within the CodeWarrior IDE, then Processor Expert automatically generates the initialization and other user support code.

The figure below illustrates the CodeWarrior IDE workspace with the Processor Expert functionally enabled showing the project manager, bean selector, error, bean inspector, and CPU Processor Expert windows. Section 6.1, “Starting a Project with Processor Expert,” provides details about configuring a CodeWarrior project to include Processor Expert.

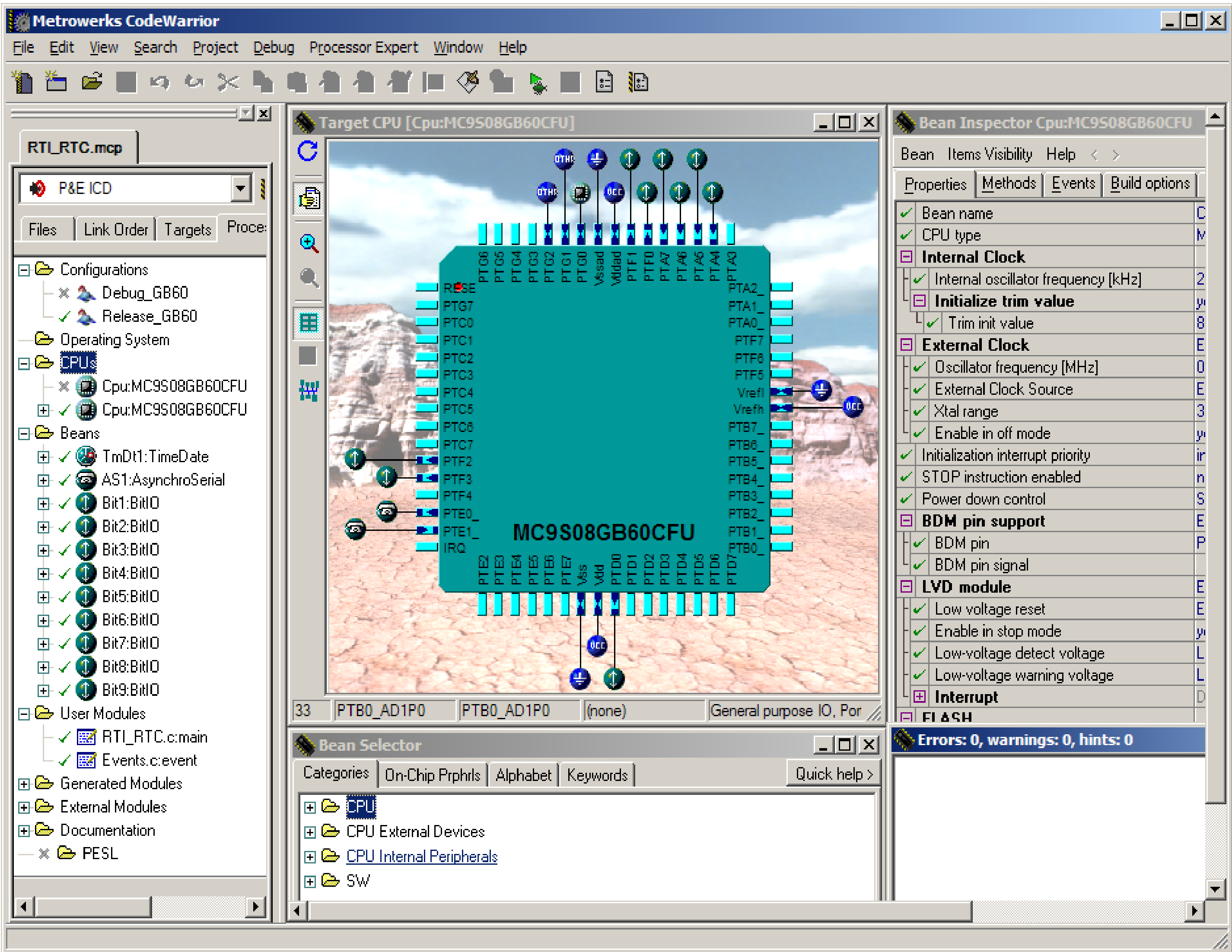


Figure 1. CodeWarrior IDE with Processor Expert Workspace

2.1 Processor Expert Benefits

Processor Expert uses an object-oriented application-building methodology using embedded beans. The embedded beads abstract the MCU hardware and register details into an intuitive software application programmer interface (API). Instead of developing software routines to initialize hardware via the MCU register map, embedded beans provide a software API and graphical interface to initialize the MCU.

In addition, an expert knowledge system is working in the background. It checks that all the MCU settings and configurations do not conflict with one another. ThProcessor Expert software API and the expert knowledge system enable an application developed in Processor Expert to be extremely portable — not only among MCU processors based on the same core platform, but also with platforms based on other Freescale MCU processors (i.e., 8/16/32/DSC). Besides the reuse benefit of using Processor Expert, other benefits are:

- Easy way to program and set-up CPU/MCU peripherals with limited knowledge about them
- Provides an interface to configure modules in real-world terms such as baud rates, instead of juggling and calculating user rate using dividers and prescalers
- Provides ready-to-use hardware drivers for peripherals
- Provide some basic software solutions such as software RTC functionality
- Ability to create new user-defined embedded beans
- Design-time settings verified by the expert knowledge system
- Allows the use of external code, libraries, and modules

2.2 What is an Embedded Bean?

Embedded beans are ready-to-use and tested building blocks for application creation. Embedded beans abstract embedded programming by providing a unified API across platforms and hiding the implementation details. That way, if and when the hardware implementation changes, the API functions are not changed. This hardware independence of the embedded beans make application portable.

The embedded beans encapsulate functionality into properties, methods, and events (this is an object oriented programming approach). More detail about these is provided here:

- **Properties** — These embedded beans' behavior attributes are defined during the application design-time and then compiled. They include MCU initialization settings such as speed of serial line, time period of the periodical interrupt, or number of channels of A/D converter. Some property settings can not change during run-time, such as memory allocations or external crystal speed.
- **Methods** — These embedded beans' behavior attributes are those that can be modified during the application runtime such as receiving serial characters, changing the SCI baud rate, or driving/reading a pin value.
- **Events** — These embedded beans' behavior attributes provide function calls when important changes happen in the bean (i.e., interrupts, received character via serial line, analog value measured, etc.)

2.3 Bean Creation and Inheritance

In general, an embedded bean can be classified as a hardware or software bean. Details about hardware and software embedded beans are provided here:

- **Hardware Beans** — Those tightly coupled with expert knowledge system and influenced by it
- **Software Beans** — These beans do not require feedback from expert knowledge system

The collection of embedded beans provided with the CodeWarrior IDE is dependant on the level of CodeWarrior IDE licensing. Some embedded beans require higher level of CodeWarrior IDE licensing. The example provided in this application note can not be developed with the special edition license; for the software RTC example, the Professional Edition of CodeWarrior is required. The Professional Edition provides access to many advanced embedded beans including the TimeData bean. Advanced embedded bean provide higher levels of functionality than beans found in the special edition license. The TimeData bean, for instance, provides fuctionality of a software RTC. Bean creation using inheritance also requires a professional license. If this project was opened using a special edition license, several licence errors would be indicated via the CodeWarrior IDE.

Inheritance refers to the creation of a new bean from an existing bean. With inheritance the new bean not only inherits existing bean functionality, but also adds additional functionality (methods, properties, or events). An example of an embedded bean is the RTC embedded bean, *TimeDate*. The *TimeDate* bean inherits functionality from the RTI-based hardware bean. This application demonstrates configuration and usage of the *TimeDate* embedded bean in Section 6.2, “Adding Embedded Beans to a Project.”

3 MC9S08GB60 and the M68DEMO908GB60

The target system for use in this application is a M68DEMO908GB60 demonstration board. This figure below shows a photo of the M68DEMO908GB60. This section lists the M68DEMO908GB60 features and provide details regarding the configuration of the M68DEMO908GB60 jumpers used for the software RTC application.

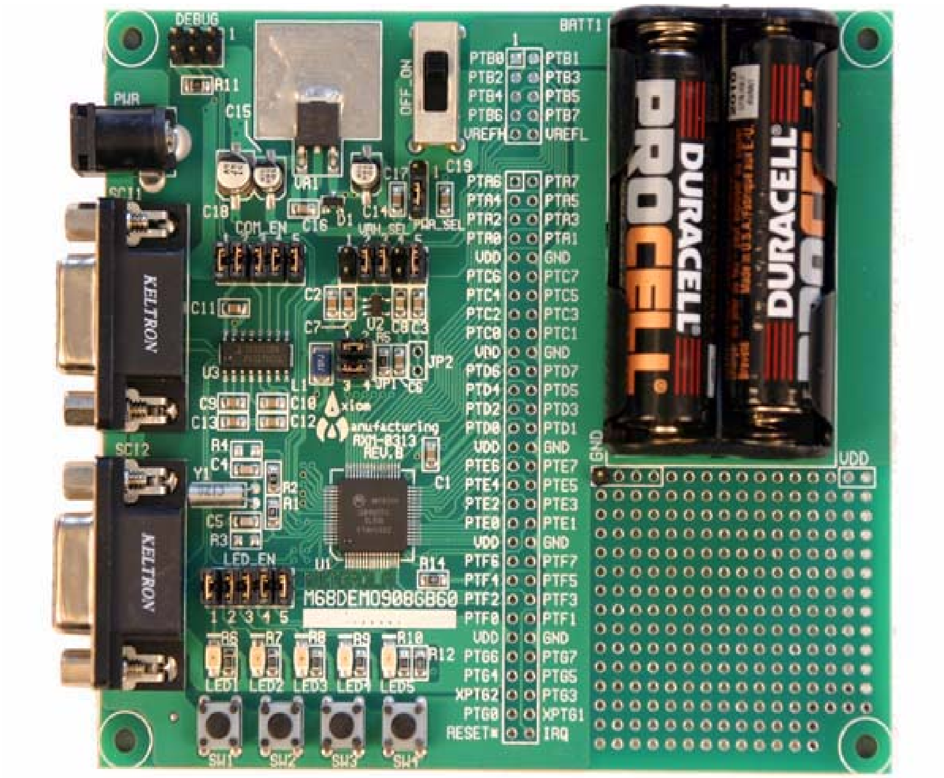


Figure 2. M68DEMO908GB60

Although an MC9S08GB60 development platform is used for this application, with only minor software/Processor Expert modifications, any HCS08 MCU could be substituted to demonstrate the software RTC.

3.1 M68DEMO908GB60 Features

The M68DEMO908GB60 can be powered using two AA batteries or an optional external power supply. It also provide the following development features:

- MC9S08GB60 MCU with 60K Flash
- 32.768 kHz external crystal
- Dual DB9 RS-232 serial ports
- Switches
- LEDs, MCU
- Pin-breakout header
- Small prototype area

3.2 M68DEMO908GB60 Configuration

The table below provides detailed jumper and switch configuration information need for proper operation of the software RTC demonstration.

Table 1. M68DEMO908GB60 Configuration

M68DEMO908GB60 Jumper/Switch	Settings
COM_EN	all jumpered
ON_OFF SWITCH	ON position
PWR_SEL	2-3 shorted when using external power
LED_EN	all jumpered
JP1	1-2 shorted 3-4 shorted
VRH_SEL	don’t care

4 RTC (Real Time Clock)

An RTC (or sometimes referred to as time of day) implementation can be either a hardware or software implementation. A hardware RTC implementation refers to one that uses an external RTC hardware module (these are sometime connected via an IIC). On the other hand, some hardware RTC implementations are provided by an on-chip peripheral in an MCU.

The primary function of an RTC implementation is to provide the time, day of the week, month, and year. The advantage of a hardware RTC is accuracy of time. Although not sought after for their accuracy, software RTCs can be a viable solution for some applications. The accuracy of the software RTC is affected by the frequency tolerance of the microcontroller clock source. If the clock source is a external crystal (for instance), a high ppm frequency tolerance would be preferred.

Software RTC can be implemented with a timer or counter that gives an interrupt based on a specified time interval. The number of time intervals are counted and then converted to time. A one second time interval is a convenient configuration for a software RTC.

Because the software RTC function is not a part of the hardware, legacy systems can implement software RTC functionality with a firmware update. Because the RTC is implemented in software, software RTCs can have a lower system cost, require fewer external components, or require less power.

5 Environment Setup

This application was developed and tested using CodeWarrior and Processor Expert running on an Windows XP PC. Version information for these tools is provided in Figure 3.

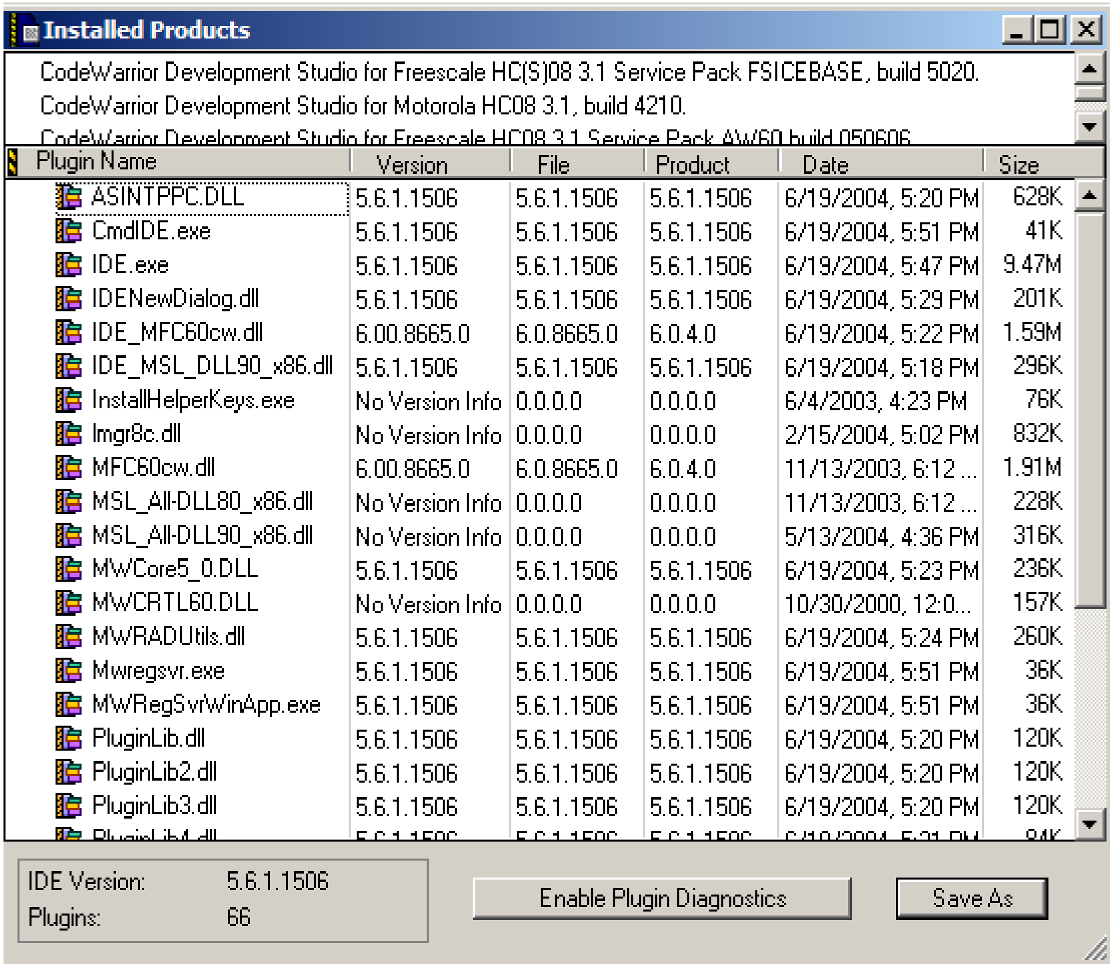


Figure 3. CodeWarrior IDE Version

Other development components include a terminal program to display SCI data via the serial port and a USB BDM pod to program the MCU. Figure 4 illustrates the connections required to program the MCU using the Code Warrior IDE and the BDM programmer.

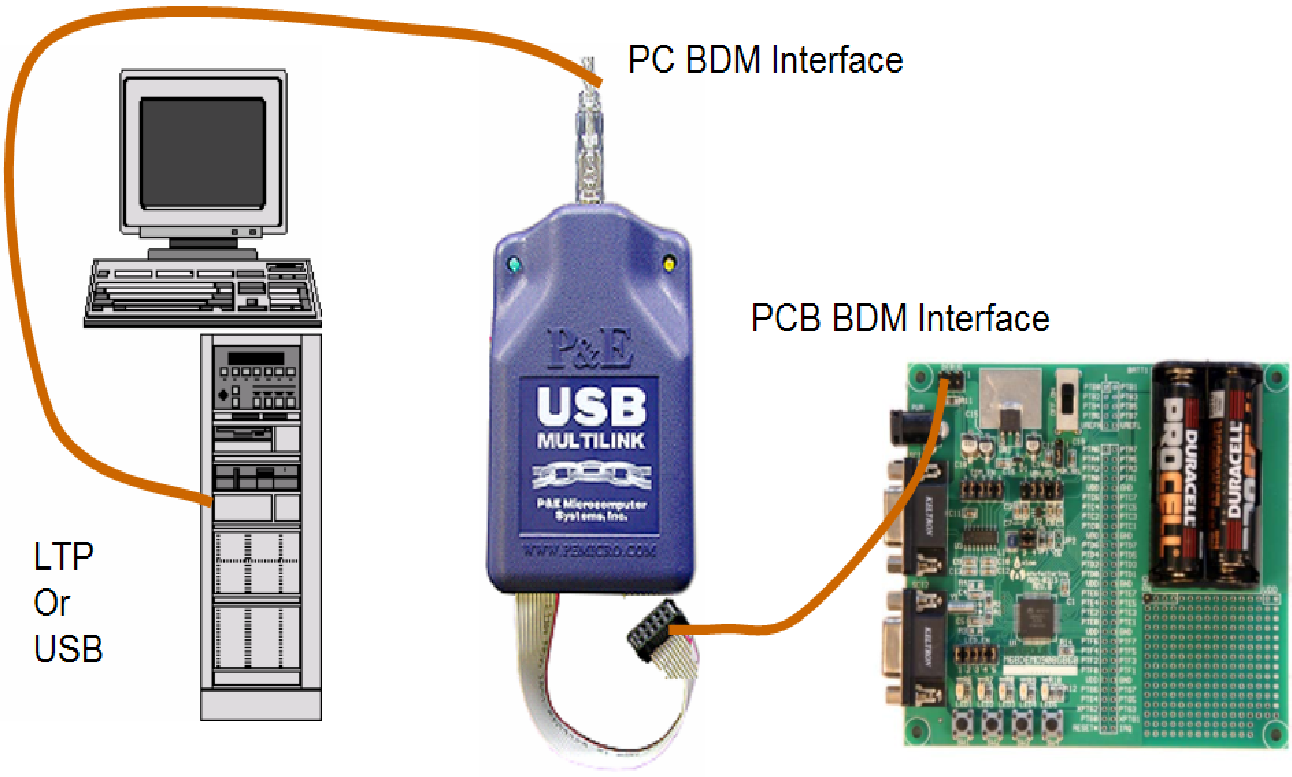


Figure 4. Development Environment Debugger/Programmer Connections

6 Project Configuration

Because the completed software is provided with this application note, the section will not detail every step of the application development. The discussion will focus on the major steps of the application development including:

- Starting a project with Processor Expert
- Add embedded beans to the project
- Resolving bean errors identified by the Processor Expert knowledge system and configuring the embedded bean properties, methods, and events
- Providing a main program
- Programming the MCU
- Demonstration of the application

6.1 Starting a Project with Processor Expert

Begin the project by opening CodeWarrior version 3.1 or later. Start a new project using the HCS08 project wizard. When the wizard asks about adding Processor Expert wizard to the project, ensure that Yes is selected as shown in Figure 5.

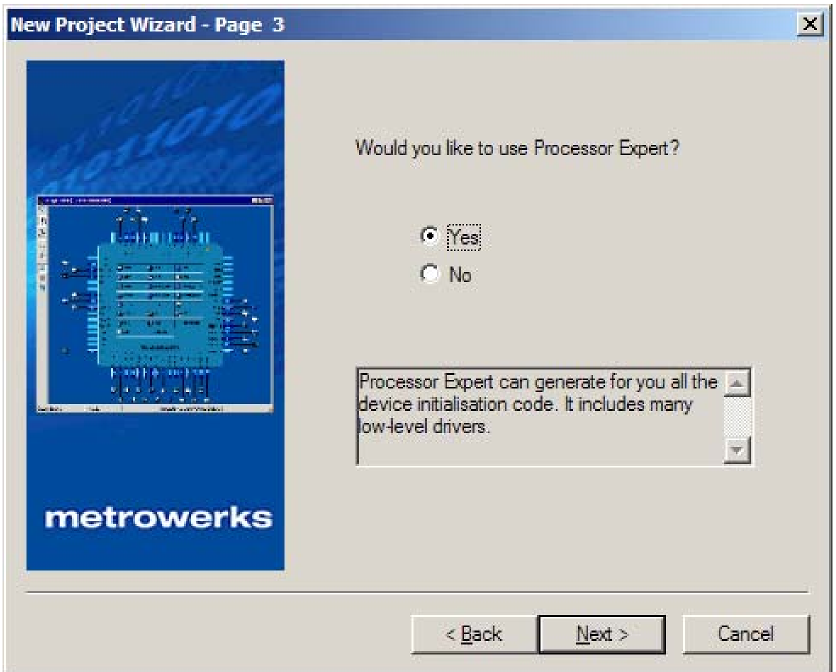


Figure 5. Project Wizard Processor Expert Option in CodeWarrior 3.1

6.2 Adding Embedded Beans to a Project

After the wizard completes, the Code Warrior IDE with the processor workspace will be opened as shown in Figure 1.

To add embedded beans, the bean selector is used. If the bean selector is not open in the IDE workspace, it can be opened via the Processor Expert menu bar. The beans selector is shown in Figure 6 with the *TimeDate* embedded bean selected. Right-click the mouse for a menu to add the *TimeDate* bean to the project.

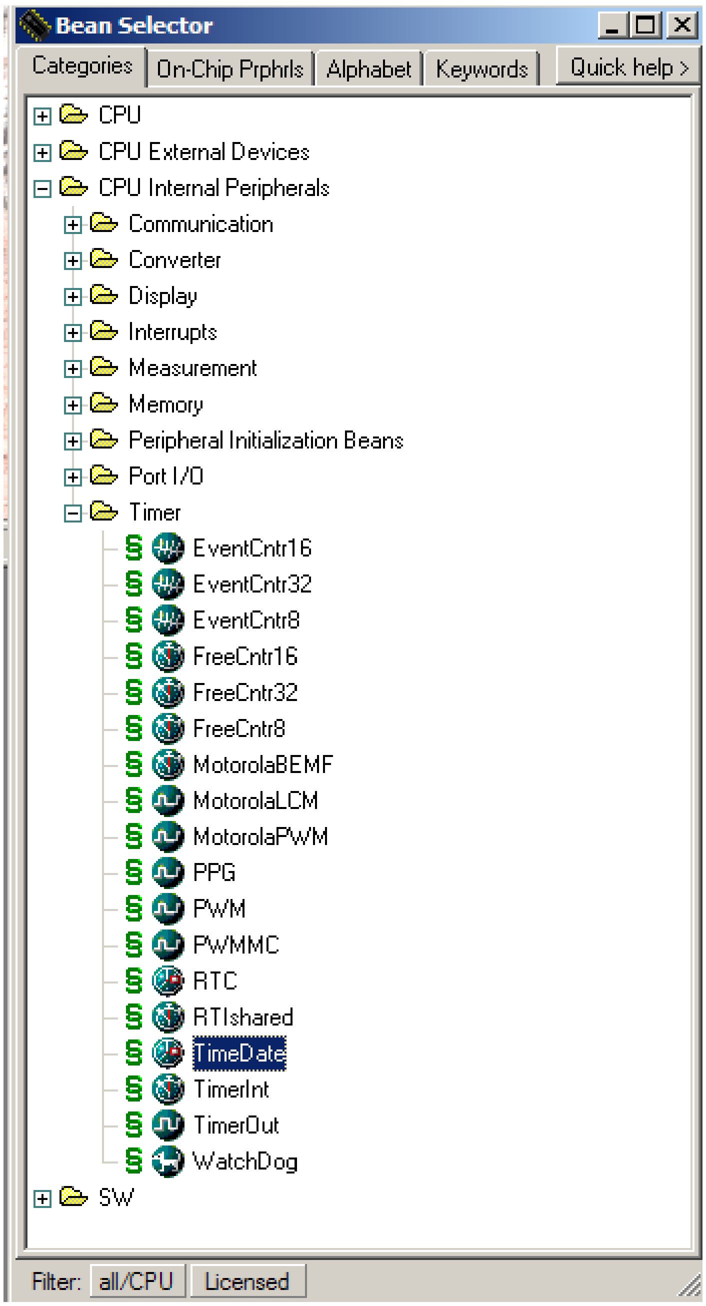


Figure 6. Bean Selector

The bean selector organizes the embedded beans in several views. Figure 6 shows the embedded bean in a categories view. The *TimeDate* bean is found in the CPU internal peripheral, timer category.

6.2.1 Project Embedded Bean Summary

Several other embedded beans are used in this application note. These embedded beans are listed in Table 2, along with the *TimeDate* embedded bean. The table list the function each bean will support, along with what MCU resource is allocated for the bean. Each one of these beans needs to be added to the project by the method described above.

Table 2. <<<Need Title>>>

Bean	Category	Function	MCU resource
TimeDate1	CPU Internal Peripheral, Timer	Software RTC	RTI
AsynchroMaster1	CPU Internal Peripheral, Communication	SCI communication used to display the time and date information to a PC terminal application	SCI0
BitIO1	CPU Internal Peripheral, Port I/O	SW1, Display Current Date information to SCI port	PTA4
BitIO2	CPU Internal Peripheral, Port I/O	SW2, Display Current Time information to SCI port	PTA5
BitIO3	CPU Internal Peripheral, Port I/O	SW3, Provide command to inverse LED1-5 display	PTA6
BitIO4	CPU Internal Peripheral, Port I/O	SW4, Provide command to blink all LEDs	PTA7
BitIO5	CPU Internal Peripheral, Port I/O	LED1 on/off control	PTF0
BitIO6	CPU Internal Peripheral, Port I/O	LED2 on/off control	PTF1
BitIO7	CPU Internal Peripheral, Port I/O	LED3 on/off control	PTF2
BitIO8	CPU Internal Peripheral, Port I/O	LED4 on/off control	PTF3
BitIO9	CPU Internal Peripheral, Port I/O	LED5 on/off control	PTD0

6.2.2 CPU Bean and Project Manager Window

Another bean that is a part of the project is the CPU bean. The bean does not need to be added because it is configured with the HCS08 project wizard when the initial project is built. Figure 7 illustrates the project manager window with the Processor Expert tab selected. This shows all the embedded beans added to the project.

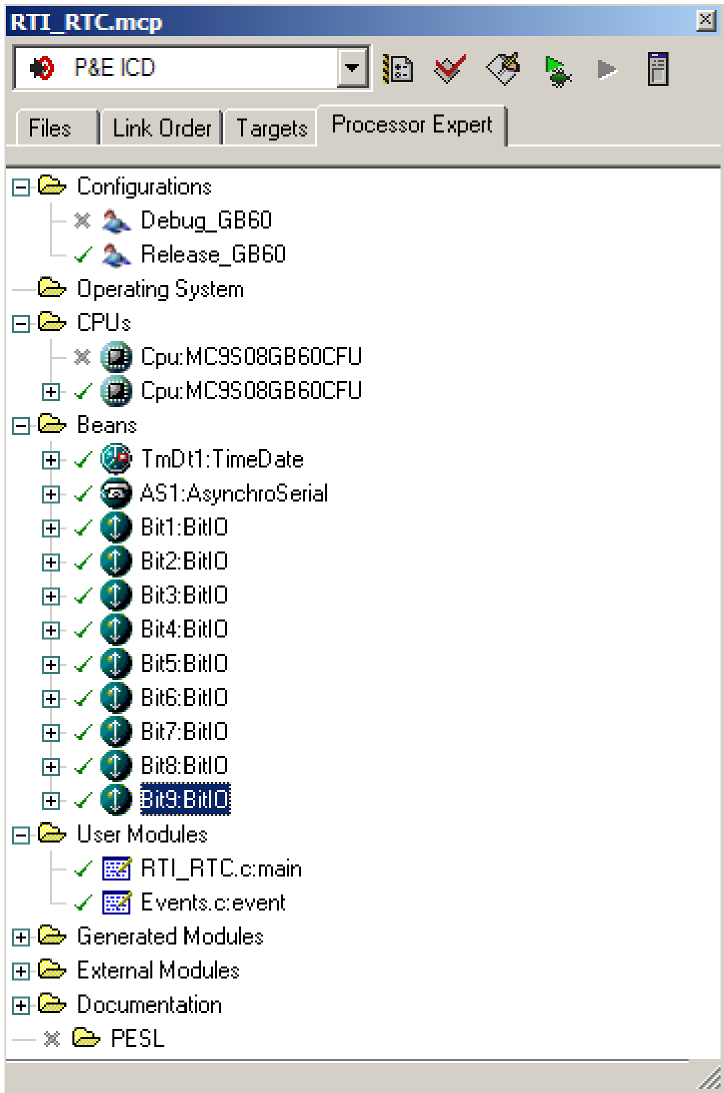


Figure 7. Project Manager Window Processor Expert View

The CPU embedded bean becomes important when porting the project to another platform. Changing the CPU bean is the first step to porting the application to another processor.

6.3 Configuring Beans with the Bean Inspector and Resolving Errors

After the embedded beans are added to the project and even before they are configured using the bean inspector, the Processor Expert knowledge system will identify system errors/conflicts and record them in the Processor Expert error window. Errors must be corrected before the Processor Expert code generation.

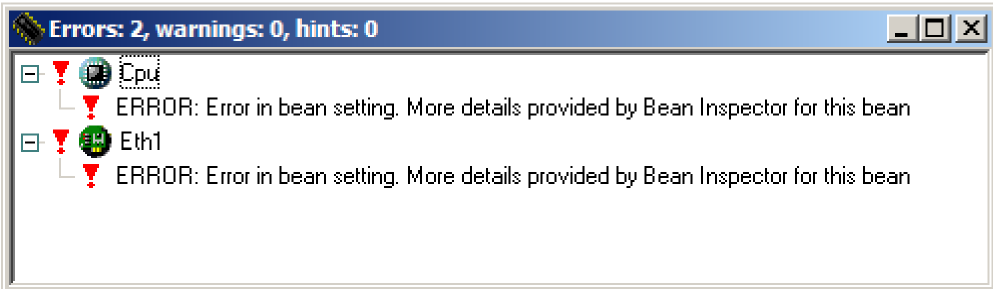


Figure 8. Processor Expert Error Window

Error identified by the Processor Expert knowledge system can include:

- Incorrect memory allocations
- Reuse of port/modules already allocated by processor expert
- Incompatible SCI baud setting based on clock configurations
- Incompatible CPU clock source / bus clock settings

6.3.1 Bean Inspector

To resolve errors and configure the embedded beans, the Processor Expert bean inspector is used. The bean inspector is a graphical user interface (GUI) provided by Processor Expert within the CodeWarrior IDE to configure the embedded bean properties, methods, and events. With the configurations made to the bean inspector, Processor Expert automatically generates the initialization and other user support code.

A figure of the *TimeDate* bean inspector is provided in Figure 9. For the *TimeDate* bean, several embedded bean property configurations are required, including:

- Indicate the software RTC timer source (Note: the RTIfree bean is used, but other timer alternatives are possible)
- Indicate the time frequency resolution is 1000 ms
- Indicate initialization values

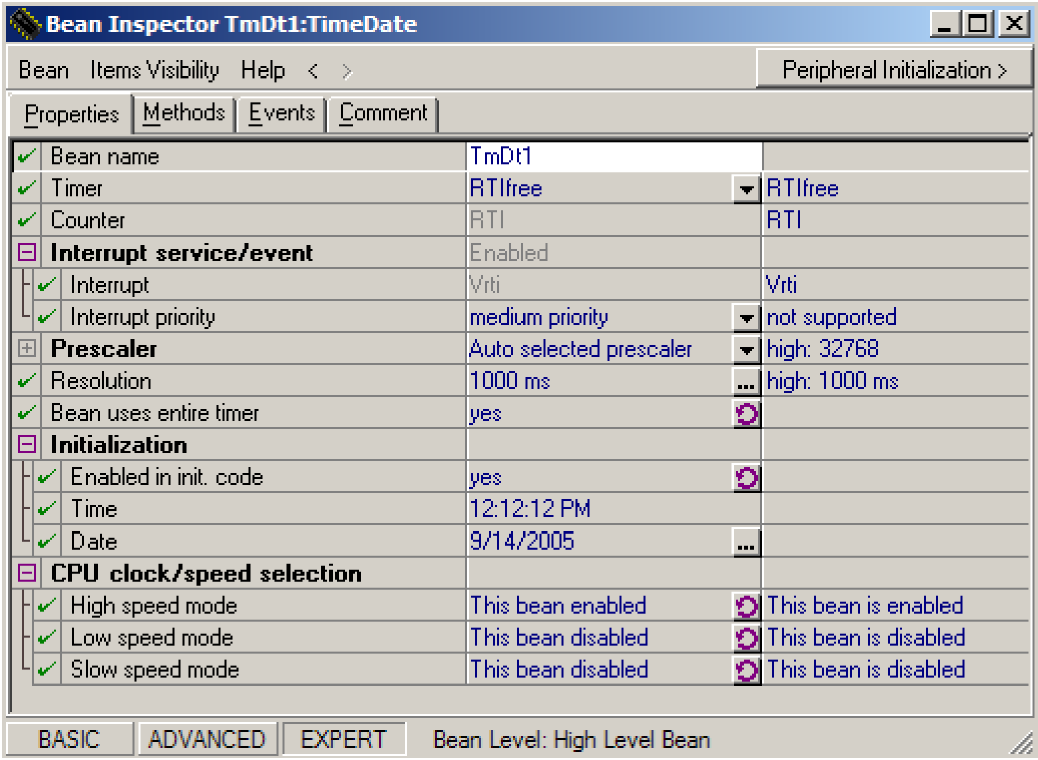


Figure 9. Bean Inspector for the *TimeDate* Bean

The bean inspector also provides an interface to configure the embedded bean methods and events (see Figure 9 dialog box tab options). Figure 10 shows the Processor Expert view of the project manger window, which lists both method and event functions. The methods are designated with an M icon and the events are designated with an E icon. In Figure 10, those method and event functions with a ✔ mark will have user code generated, while thos with an ✘ mark will not. Accessing the methods and events tab view of the bean inspector, the user can select which method and event functions are enabled for code generation.

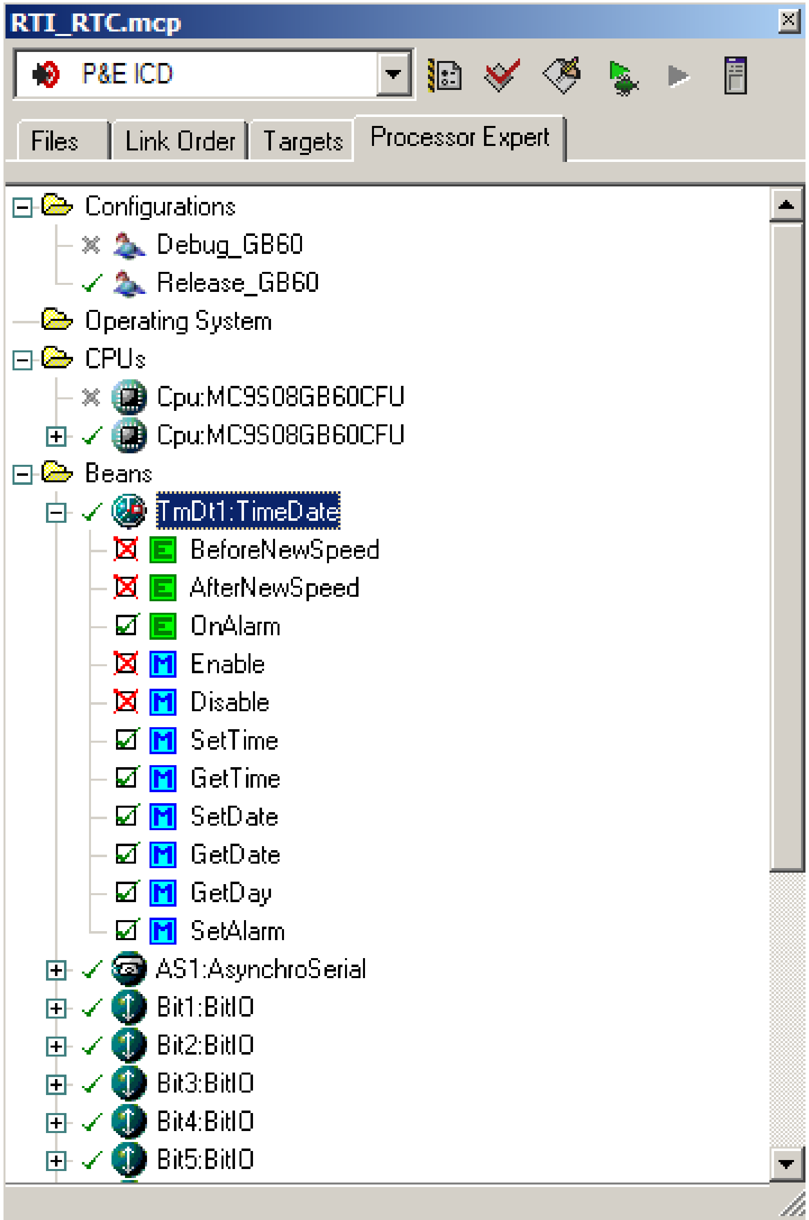


Figure 10. Project Manger Window Showing Methods and Events

6.3.2 Embedded Bean Help

Every embedded bean property, method, and event is documented. A help html page can be opened from the Processor Expert view of the CodeWarrior project manager window. To open the help for a particular embedded bean, right-click the embedded bean and select Help in the menu as shown in Figure 11. The embedded bean help window also shows example code for each embedded bean.

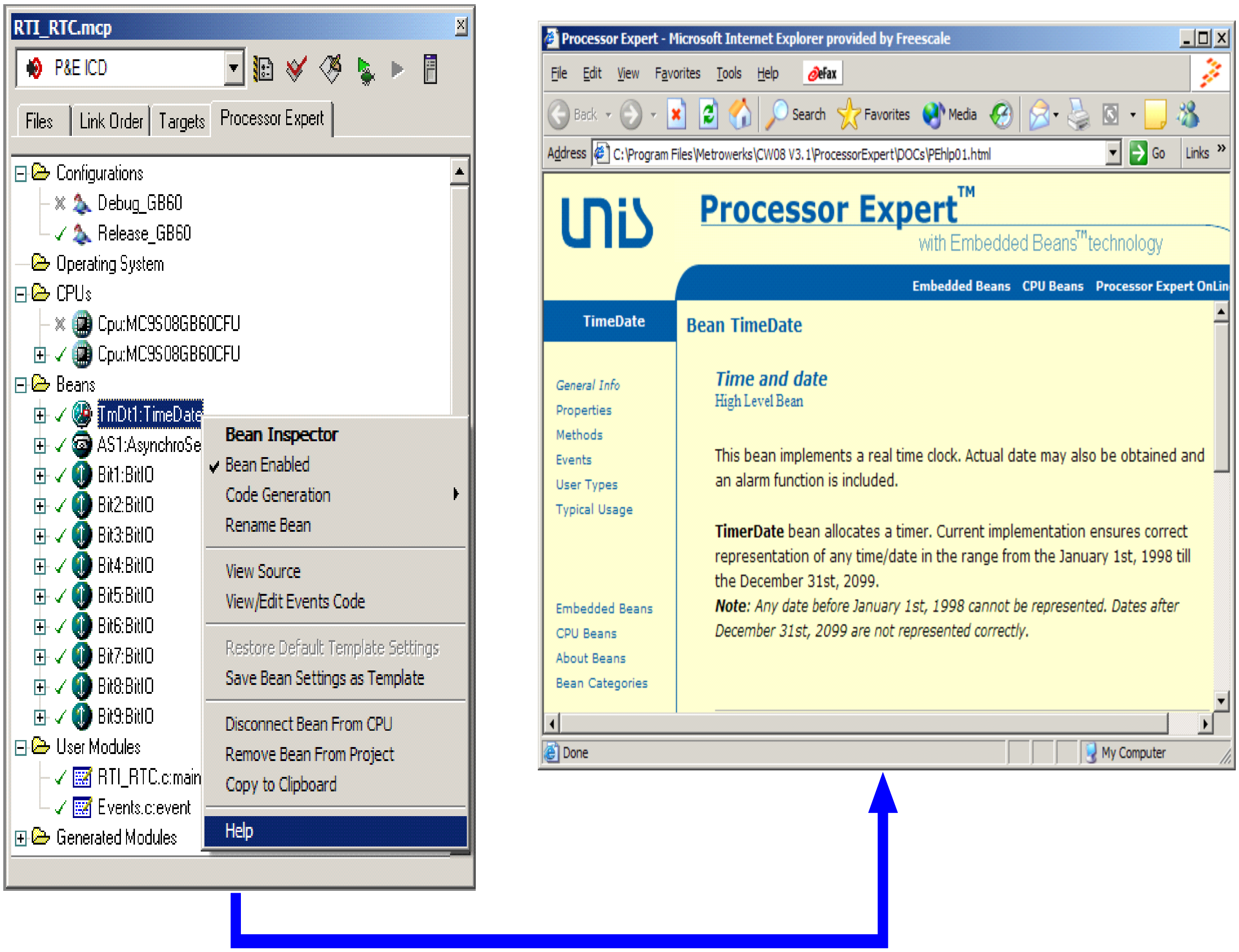


Figure 11. TimeDate Embedded Bean Help HTML Page

6.3.3 Summary of Embedded Bean Configuration for Software RTC

Table 3 itemizes the minimum Processor Expert settings the user must configure for each embedded bean for the software RTC application. Embedded beans properties, methods, and events are configured using the bean inspector, as detailed in the discussion above.

Table 3. Embedded Bean Configuration Settings

Bean	Function	MCU resource	Bean Property Configuration Settings
CPU1	CPU	CPU	• Indicate the 32.768 kHz external clock • Indicate a bus clock frequency • Indicate any PRM file build options
TimeDate1	Software RTC	RTI	• Indicate the software RTC timer source (Note: the RTIfree bean is used, but other timer alternatives are possible) • Indicate the time frequency resolution (1000 ms) • Indicate initialization values for time and date
AsynchroMaster1	SCI communication	SCI1	• Indicate which SCI channel is used for communication • Indicate a baud rate (115,200 bps)
BitIO1	SW1	PTA4	• Allocate a pin for the I/O • Indicate a pin direction — Input • Indicate a pull resistor — Pullup
BitIO2	SW2	PTA5	• Allocate a pin for the I/O • Indicate a pin direction — Input • Indicate a pull resistor — Pullup
BitIO3	SW3	PTA6	• Allocate a pin for the I/O • Indicate a pin direction — Input • Indicate a pull resistor — Pullup
BitIO4	SW4	PTA7	• Allocate a pin for the I/O • Indicate a pin direction — Input • Indicate a pull resistor — Pullup
BitIO5	LED1	PTF0	• Allocate a pin for the I/O • Indicate a pin direction — Output
BitIO6	LED2	PTF1	• Allocate a pin for the I/O • Indicate a pin direction — Output
BitIO7	LED3	PTF2	• Allocate a pin for the I/O • Indicate a pin direction — Output
BitIO8	LED4	PTF3	• Allocate a pin for the I/O • Indicate a pin direction — Output
BitIO9	LED5	PTD0	• Allocate a pin for the I/O • Indicate a pin direction — Output

6.4 Generate Processor Expert Code

After all the Processor Expert errors have been resolved and the embedded bean are configured correctly using the bean inspector, the Processor Expert generate code command can be executed. No additional code in main is required to generate the Processor Expert generated code. The generate code command is accessible via the IDE manu bar as shown in Figure 12.

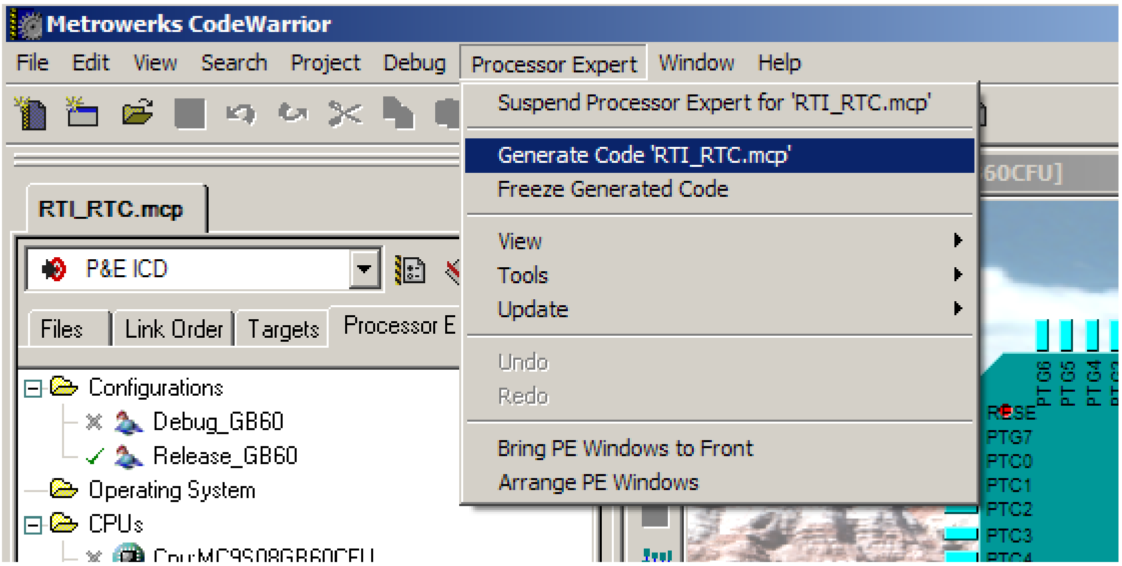


Figure 12. Processor Expert Generate Code Command

Figure 13 shows the project manager with the files tab selected. The files view shows both the Processor Expert generated code group and the user modules code group. The files in the Processor Expert generated code group should never be edited by the user. These are strictly maintained by Processor Expert and the Processor Expert knowledge system.

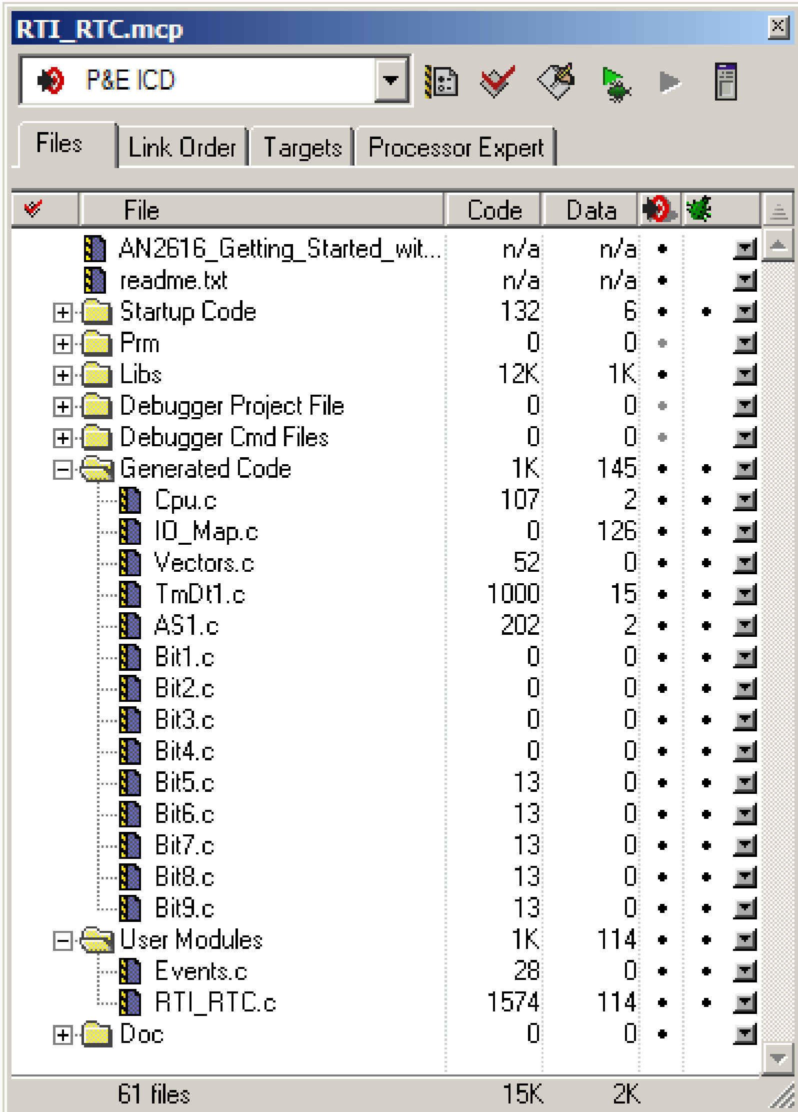


Figure 13. Project Manager Window Showing Processor EXpert Generated Code

6.4.1 TmDt1_Interrupt()

The code below is an example of the code that was generate by Processor Expert. This code is called by the periodic interrupts of the real time interrupt (RTI) module which was configured by the bean inspector to interrupt every second. The vector table is found in vector.c in the generated code code group. The *TmDt1_Interrupt()* function is found in the TmDt1.c file.

```
/*
** ===================================================================
**     Method      :  TmDt1_Interrupt (bean TimeDate)
**
**     Description :
**         This method is internal. It is used by Processor Expert
**         only.
** ===================================================================
*/
__interrupt void TmDt1_Interrupt(void)
{
```

```
  const byte * ptr;                    /* Pointer to ULY/LY table */

  SRTISC_RTIACK = 1;                   /* Reset real-time interrupt request flag */
  TotalHthH += 100;                    /* Software timer counter increment by timer period (10 ms) */
  if (TotalHthH >= 8640000) {          /* Does the counter reach 24 hours? */
    TotalHthH -= 8640000;              /* If yes then reset it by subtracting exactly 24 hours */
    AlarmFlg = FALSE;                  /* Reset alarm flag - alarm has not occured during these 24 hours yet */
    CntDOW++;                          /* Increment Sun - Sat counter */
    if (CntDOW >= 7)                   /* Sun - Sat counter overflow? */
      CntDOW = 0;                      /* Set Sun - Sat counter on Mon */
    CntDay++;                          /* Increment day counter */
    if (CntYear & 3)                   /* Is this year un-leap-one? */
      ptr = ULY;                       /* Set pointer to un-leap-year day table */
    else                               /* Is this year leap-one? */
      ptr = LY;                        /* Set pointer to leap-year day table */
    ptr--;                             /* Decrement the pointer */
    if (CntDay > ptr[CntMonth]) {      /* Day counter overflow? */
      CntDay = 1;                      /* Set day counter on 1 */
      CntMonth++;                      /* Increment month counter */
      if (CntMonth > 12) {             /* Month counter overflow? */
        CntMonth = 1;                  /* Set month counter on 1 */
        CntYear++;                     /* Increment year counter */
      }
    }
  }
  if (!AlarmFlg) {                     /* Has the alarm already been on? */
    if (TotalHthH >= AlarmHth) {       /* Is the condition for alarm invocation satisfied? */
      AlarmFlg = TRUE;                 /* Set alarm flag - alarm has been invocated */
      TmDt1_OnAlarm();                 /* Invoke user event */
    }
  }
}

/* END TmDt1. */
```

6.4.2 TmDt1_SetDate()

The ***TimeDate*** embedded bean also manages and generates all code needed to set and get the date and time. The user does not have to develop code that converts a count of the RTI interrupts into more conventional date and time variables in the format of MM/DD/YYYY and HH:MM:SS, respectively. The *TimerDate* bean current implementation ensures correct representation of time and date in the range from the January 1st, 1998, until December 31st, 2099. The source code below is provided for the *TmDt1_SetDate()TimeDate* function. This code is automatically generated by Processor Expert and must not be edited.

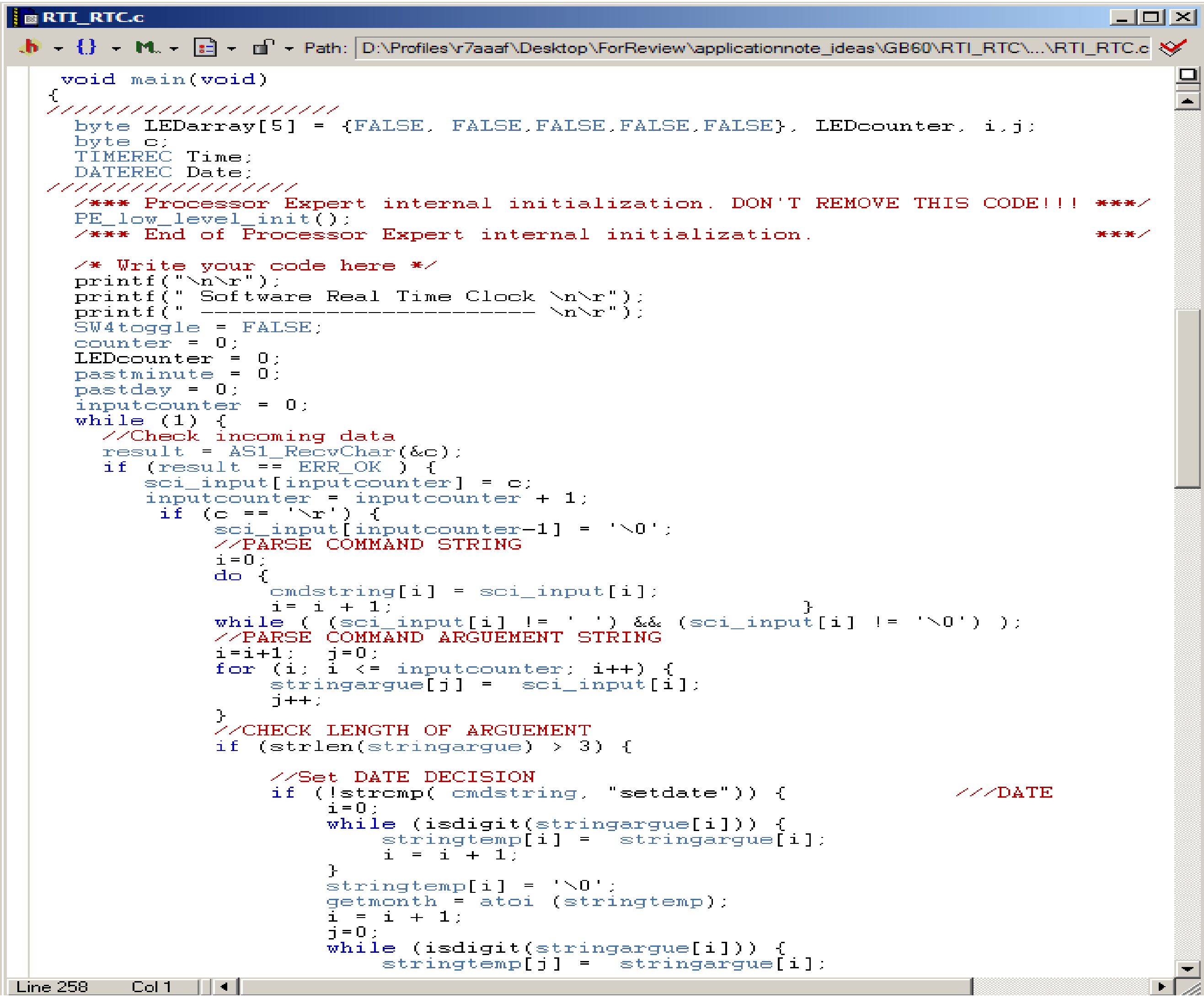
```
/*
** ===================================================================
**     Method      :  TmDt1_SetDate (bean TimeDate)
**
**     Description :
**         Set a new actual date.
**     Parameters  :
**         NAME            - DESCRIPTION
**         Year            - Years (16-bit unsigned integer)
**         Month           - Months (8-bit unsigned integer)
**         Day             - Days (8-bit unsigned integer)
**     Returns     :
**         ---             - Error code, possible codes:
**                           ERR_OK - OK
**                           ERR_SPEED - This device does not work in
**                           the active speed mode
**                           ERR_RANGE - Parameter out of range
** ===================================================================
*/
byte TmDt1_SetDate(word Year,byte Month,byte Day)
{
```

```
  word tY = 1998;                     /* Year counter, starting with 1998 */
  byte tM = 1;                        /* Month counter, starting with January */
  byte tD = 1;                        /* Day counter, starting with 1 */
  byte tW = 4;                        /* Sun - Sat counter, starting with Thu */
  const byte * ptr;                   /* Pointer to ULY/LY table */

  if ((Year < 1998) || (Year > 2099) || (Month > 12) || (Month == 0) || (Day > 31) || (Day == 0)) /* Test correctness of
given parameters */
    return ERR_RANGE;                 /* If not correct then error */
  if (tY & 3)                         /* Is given year un-leap-one? */
    ptr = ULY;                        /* Set pointer to un-leap-year day table */
  else                                /* Is given year leap-one? */
    ptr = LY;                         /* Set pointer to leap-year day table */
  ptr--;                              /* Decrement pointer */
  for (;;) {
    if ((Year == tY) && (Month == tM)) { /* Is year and month equal with given parameters? */
      if (ptr[tM] < Day)             /* Does the obtained number of days exceed number of days in the appropriate month
& year? */
        return ERR_RANGE;             /* If yes (incorrect date inserted) then error */
      if (tD == Day)                  /* Does the day match the given one? */
        break;                        /* If yes then date inserted correctly */
    }
    tW++;                             /* Increment Sun - Sat counter */
    if (tW >= 7)                      /* Sun - Sat counter overflow? */
      tW = 0;                         /* Set Sun - Sat counter on Mon */
    tD++;                             /* Increment day counter */
    if (tD > ptr[tM]) {               /* Day counter overflow? */
      tD = 1;                         /* Set day counter on 1 */
      tM++;                           /* Increment month counter */
      if (tM > 12) {                  /* Month counter overflow? */
        tM = 1;                       /* Set month counter on 1 */
        tY++;                         /* Increment year counter */
        if (tY & 3)                   /* Is this year un-leap-one? */
          ptr = ULY;                  /* Set pointer to un-leap-year day table */
        else                          /* Is this year leap-one? */
          ptr = LY;                   /* Set pointer to leap-year day table */
        ptr--;                        /* Decrement pointer */
      }
    }
  }
  EnterCritical();                    /* Save the PS register */
  CntDOW = tW;                        /* Set Sun - Sat counter to calculated value of day in a week */
  CntDay = tD;                        /* Set day counter to the given value */
  CntMonth = tM;                      /* Set month counter to the given value */
  CntYear = tY;                       /* Set year counter to the given value */
  ExitCritical();                     /* Restore the PS register */
  return ERR_OK;                      /* OK */
}
```

6.5 Providing a Main Program

For any application, the user must add code to *main()* and using Processor Expert does not change this requirement. What does change is that the user can start writing the application code because the MCU initialization and peripheral driver codes have been generated by Processor Expert. The MCU initialization code generated by Processor Expert is called by the *PE_low_level_init()* function. The function is found in the Cpu.c file in the generated code code group. Figure 14 provides a partial listing of *main()* for the software RTC application.




```
void main(void)
{
//////////////////////////
  byte LEDarray[5] = {FALSE, FALSE,FALSE,FALSE,FALSE}, LEDcounter, i,j;
  byte c;
  TIMEREC Time;
  DATEREC Date;
//////////////////////
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  printf("\n\r");
  printf(" Software Real Time Clock \n\r");
  printf(" ------------------------ \n\r");
  SW4toggle = FALSE;
  counter = 0;
  LEDcounter = 0;
  pastminute = 0;
  pastday = 0;
  inputcounter = 0;
  while (1) {
    //Check incoming data
    result = AS1_RecvChar(&c);
    if (result == ERR_OK ) {
       sci_input[inputcounter] = c;
       inputcounter = inputcounter + 1;
        if (c == '\r') {
           sci_input[inputcounter-1] = '\0';
           //PARSE COMMAND STRING
           i=0;
           do {
               cmdstring[i] = sci_input[i];
               i= i + 1;                                  }
           while ( (sci_input[i] != ' ') && (sci_input[i] != '\0') );
           //PARSE COMMAND ARGUEMENT STRING
           i=i+1;   j=0;
           for (i; i <= inputcounter; i++) {
               stringargue[j] =  sci_input[i];
               j++;
           }
           //CHECK LENGTH OF ARGUEMENT
           if (strlen(stringargue) > 3) {

               //Set DATE DECISION
               if (!strcmp( cmdstring, "setdate")) {              ///DATE
                   i=0;
                   while (isdigit(stringargue[i])) {
                       stringtemp[i] =  stringargue[i];
                       i = i + 1;
                   }
                   stringtemp[i] = '\0';
                   getmonth = atoi (stringtemp);
                   i = i + 1;
                   j=0;
                   while (isdigit(stringargue[i])) {
                       stringtemp[j] =  stringargue[i];
```

Figure 14. Partial Listing of main() for the Software RTC Application

The complete source code for *main()* and the complete project is provided as an attachment to this application note for reference, so only an overview of the application code is provided here. The overview will include a summary of the software RTC application functionality and a listing of the Processor Expert functions used.

6.5.1 Software RTC Application Details

The application primarily demonstrates a software RTC, but there is also additional functionality to provide serial communication, button/switch functions, and LED operation.

The time and date calculations are completely managed by the *TimeDate* embedded bean's properties, methods, and events. For the application to get or configure the date or time, it must call the functions of the *TimeDate* API. In the application, the time and date results are transmitted via the SCI so that they can be displayed via a terminal program in Windows. The MCU uses an SCI baud rate of 115,200 bps that was specified in the *AsynchroMaster* bean inspector.

The terminal program is used also to capture user input so the time and date can be changed. The user input is received by the MCU SCI peripheral and a command processor is used to determine and execute user time and date changes. *Main()* loops forever, collecting characters from the SCI into a command buffer and does not process the user command until a carriage return character is received.

The application also uses the LED1-5 and SW1-4 on the GB60 DEMO board. SW1 and SW2 force the current time and date to the SCI. SW3 and SW4 provide control of LEDs 1 through 5. Figure 15 shows a simplified flow chart for *main()*.

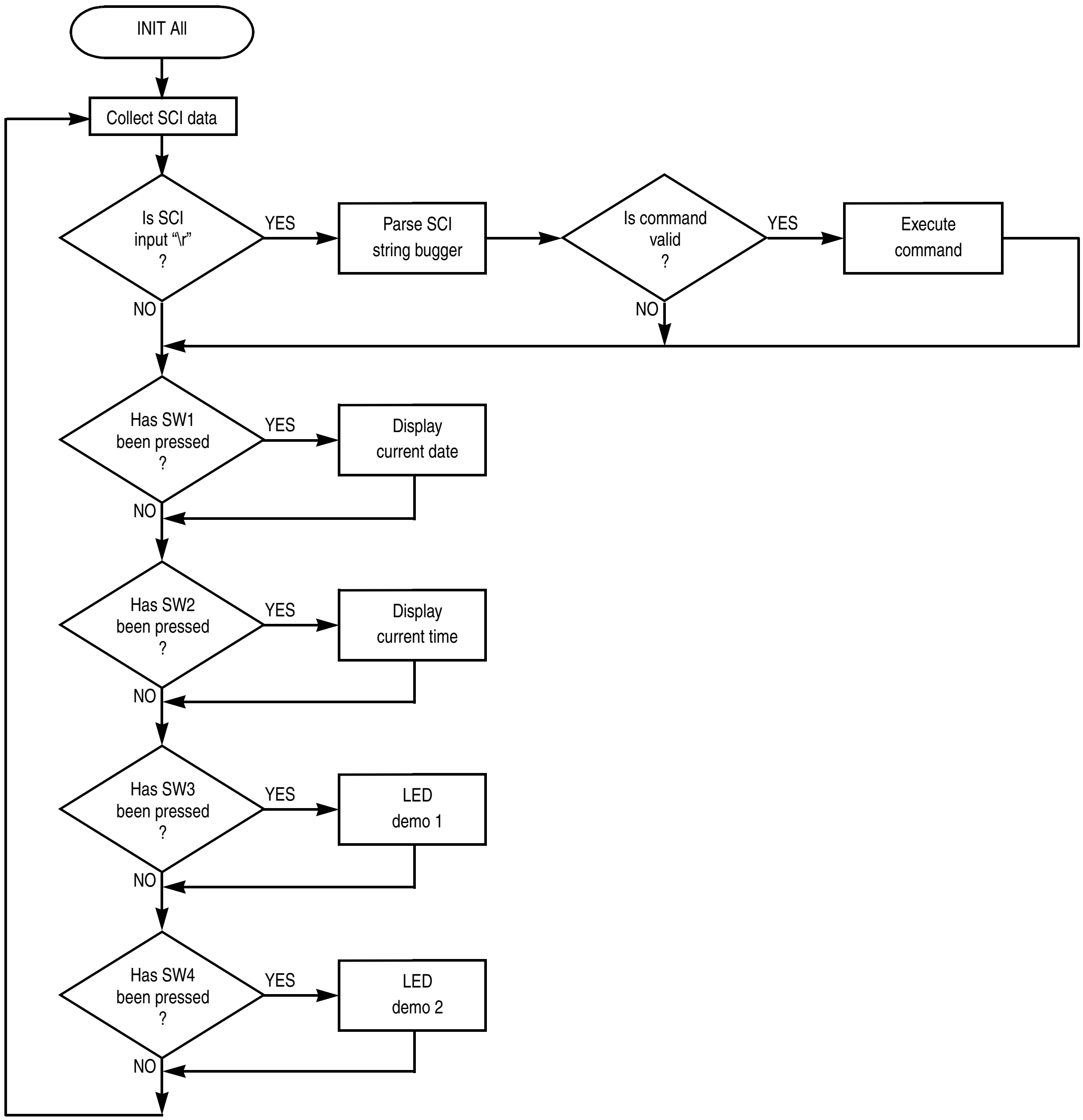


Figure 15. Main Flowchart

6.5.2 Processor Expert Functions Used

The table below itemizes the Processor Expert functions used in this software RTC application. The table also provides an overview of each function.

Table 4. Overview of Selected Processor Expert Functions

Embedded Bean	Generate Code File	Functions	Description for Bean Help
Cpu1	*Cpu.c*	*PE_low_level_Init*	Configures the peripheral base on input to the bean inspector. Calls *init* function of other embedded beans
TimeDate1	*TmDt1.c*	*TmDt1_SetDate*	Sets a new date
		TmDt1_SetTime	Sets a new time
		TmDt1_SetAlarm	*SetAlarm* — Sets a new time of alarm. (only time, not date — alarm event *OnAlarm* is called every 24 hours). Setting time of alarm out of 24 hour interval disables its function.
		TmDt1_GetDate	Gets the current date
		TmDt1_GetTime	Gets the current time
AsynchroMaster1	*AS1.c*	*AS1_SendChar*	*SendChar* — Send one character to the channel. If the bean is temporarily disabled (Disable method) SendChar method stores data only into output buffer. In case of zero output buffer size, only one character can be stored. Enabling the bean (Enable method) starts transmission of stored data. This method is available only if the transmitter property is enabled.
		AS1_RecvChar	*RecvChar* — If any data received, this method returns one character, otherwise it returns error code (it does not wait for data). This method is enabled only if the receiver property is enabled.
BitIO1to4	*BitN.c*	*BitN_GetVal*	*GetVal* — Returns the value of the *Input/Output* bean. If the direction is input, then the input value of the pin is read and returned. If the direction is output, then the last written value is returned.
BitIO5to9	*BitN.c*	*BitN_GetVal*	*GetVal* — Returns the value of the *Input/Output* bean. If the direction is input, then the input value of the pin is read and returned. If the direction is output, then the last written value is returned.
		BitN_PutVal	*PutVal* — Specified value is passed to the *Input/Output* bean. If the direction is input, saves the value to a memory or a register, this value will be written to the pin after switching to the output mode (using *SetDir(TRUE)*). If the direction is output, it writes the value to the pin. (Method is available only if the direction = output or input/output).

6.6 Programming the MCU Target

After the main code is added, the project code can be downloaded into the target MCU Flash memory. In this project, the USB multilink pod was specified as the programmer/debugger target (e.g., P&E ICD). Figure 16 shows pressing the debug icon will initiate the programming of the target MCU Flash memory.

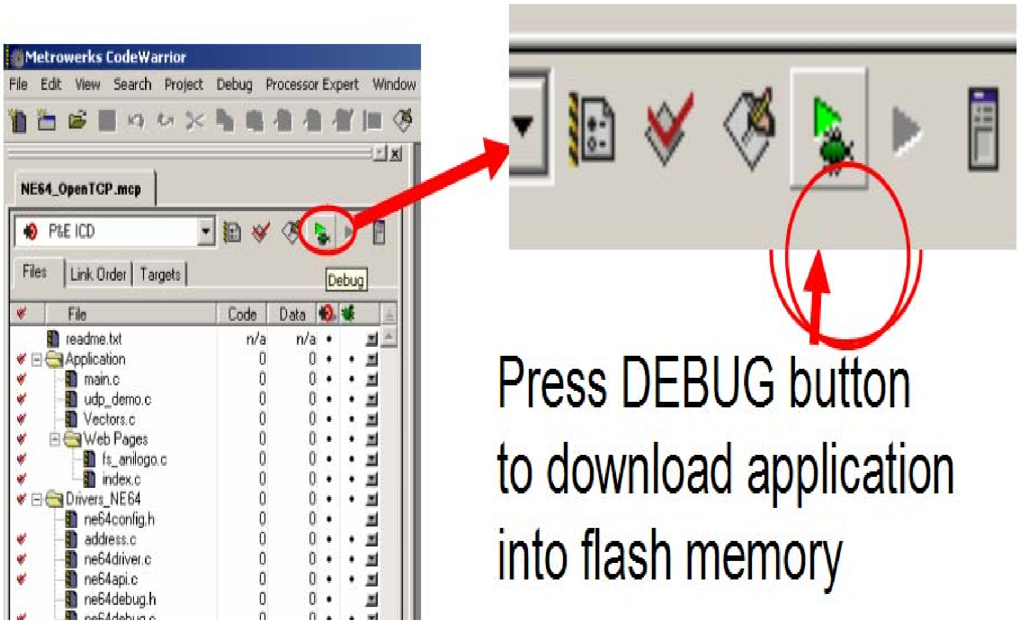


Figure 16. Code Warrior IDE Debug Icon

Figure 17 illustrates the HiWave programmer/debugger program that opens for Flash programming. The RUN icon executes the software RTC application code.

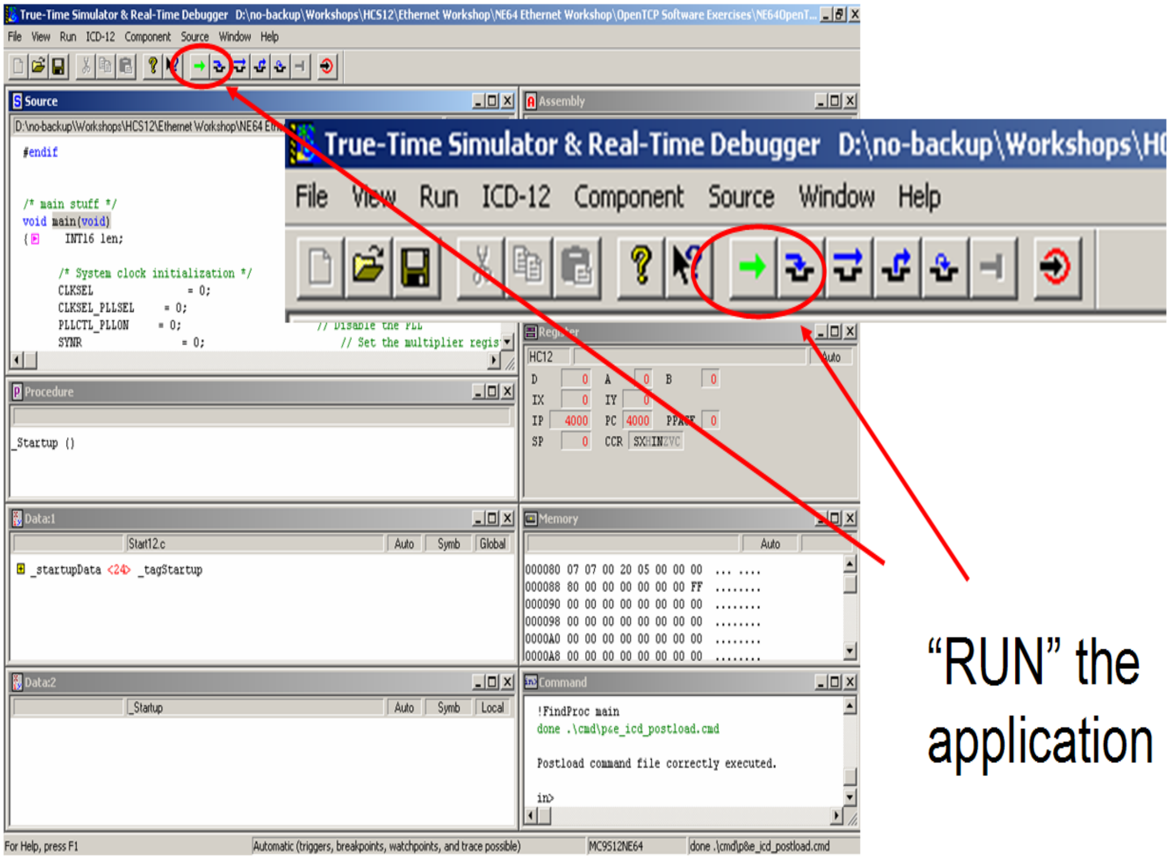


Figure 17. CodeWarrior Programmer/Debugger Interface

6.6.1 Demonstrating the Software RTC Application

As discussed above, a terminal program is used to capture the MCU's SCI output at baud rate of 115,200 bps. The firmware *main()* program updates the time display automatically every minute, yet it will provide an automatic display only of the current date when the day field changes. The date and time can be forced to the display by pressing SW1 and SW2 buttons (SW3 and SW4 buttons control LED 1 through 5). Figure 18 shows the software RTC date and time output to a terminal program.

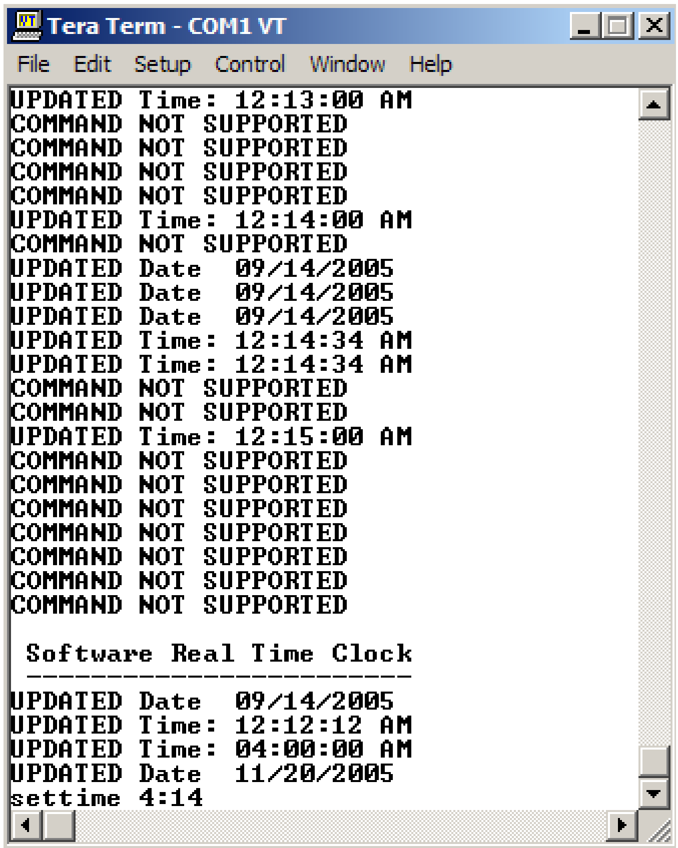


Figure 18. Time Display via the Serial Port

The terminal program is also used to update the time, date, and alarm information using the firmware supported commands listed below. The syntax for these commands is also provided.

- settime (syntax: settime HH:MM:SS)
- setdate (syntax: setdate DD:MM:YYYY)
- setalarm (syntax: setalarm HH:MM:SS)

7 Conclusion

Using Processor Expert and embedded beans can facilitate faster application development. There may be, however, trade-offs.

- Instead of register fields, the developer must learn and understand the embedded bean API (properties, methods, and event functions)
- The embedded bean API may not meet the application requirements, but, using embedded bean creation and inheritance, one can possibly mitigate this issue
- The user loses direct register control/interaction
- The Processor Expert generated code may be completing tasks in a pre-defined sequence and may reduce flexibility
- Generated code is controlled by the IDE and must not be modified.
- Using Processor Expert may require additional licensing to gain access to all embedded beans and the embedded bean creation functionality

Using Processor Expert shortens development effort and time. With the Processor Expert generated code, less time was required to develop initialization and peripheral driver code. In fact, to configure the peripheral, we did not even need to know what MCU registers were involved or how to set them up.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN2949
Rev. 1, 1/2006