

# POINTERS

Winter/Spring 1999

Let's Embed the World!

## Keil Ships C166 Version 4 and $\mu$ Vision2

Keil Software is now shipping C166 Version 4 with  $\mu$ Vision2. This release of the Keil C166, C167, and ST10 development tools offers a number of new enhancements and features.

$\mu$ Vision2 has been completely redesigned and now incorporates the IDE and debugger all in a single application.  $\mu$ Vision2 was designed using requests and suggestions we have received from customers over the past two years. Following is a brief list of the new features and benefits:

- The Project Manager now lets you group files and targets within a project. You can create multiple target programs from a single project file. For example, you may have one target for debugging with MON51 and another target for creating an Intel HEX file for programming an EPROM.
- A Device Database makes it easy to get started with a particular microcontroller or evaluation board. The database includes all the necessary options for numerous different chips, emulators, and monitor configurations.

- Tool Options may now be set for either a target, a group of files, or an individual source file. This provides greater flexibility for complex projects.
- A new Locals page has been added to the Watch Window. This page displays all local variables for a function.
- The integrated Make facility assembles, compiles, and links your source files and displays errors and warnings in real-time. You may correct errors while the project compiles in the background.
- A Project-wide Search feature lets you locate each occurrence of a text string in your source files.
- Multiple views are now available for the Memory Window and the Serial Window.
- A Books tab provides instant access to PDF data sheets, User's Guides, and on-line manuals.

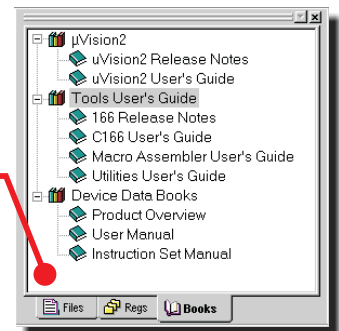
You may contact our sales department for a FREE Evaluation CD-ROM and Quick Start guide. A 4K-limited compiler lets you evaluate C166 Version 4 and  $\mu$ Vision2.

If you can't wait for the CD, you may download a demo from the Keil web site.

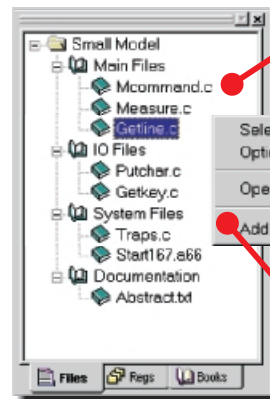
<http://www.keil.com/demo>

Upgrades are free for users who have purchased a C166 Version 3 Kit on or after January 1, 1998.

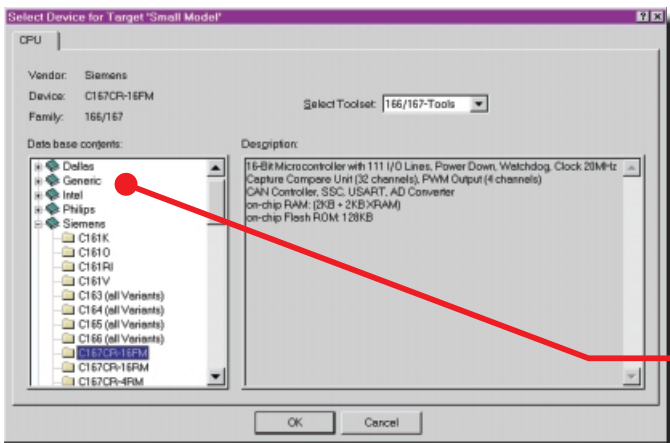
Contact our sales department for more information about product prices and upgrades.



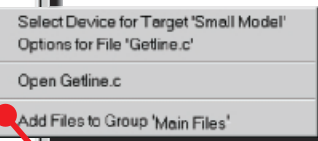
The Project Window's Books tab lists the on-line manuals that are available.



The Files tab in the Project window displays the currently selected target along with groups and files. You may drag & drop files within this window.



Generic devices may be used if a specific device is not already in the database.



The local menu opens when you click the right mouse button. Here, you may set options for targets, groups, and files.

### See inside for information about...

- Controller Area Network (CAN)
- RTX51 CAN Library
- Siemens C505C CAN Support
- C51 Function Pointers
- MCB517 Evaluation Boards

## MCB517 and RTX51 Make CAN Easy

The Keil MCB517 Evaluation Board includes an on-board CAN controller (the Siemens 81C90) that is easy to configure. RTX51 includes a CAN library that supports the 81C90 and numerous other CAN controllers. Together, these two tools let you experiment with CAN and get a network up and running very quickly.

The RTX51 CAN library includes calls to initialize the CAN controller, send messages, and receive messages.

To begin a CAN program, you must create the CAN task and initialize the CAN controller. For example:

```
can_task_create ();
can_hw_init (
    0x23, 0x42, 0xF8, 0x00, 0x04);
```

The arguments to `can_hw_init` depend on the controller used. They configure bit rate, bus timing, and so on.

Once the CAN controller is initialized, you may create message objects to send or receive. For example:

```
can_def_obj (0x0123, 6, D_SEND);
can_def_obj (0x0234, 1, D_REC);
```

These statements define a transmit object 6 bytes long with ID 0x0123 and a receive object 1 byte long with ID 0x0234.

Finally, you are ready to let the controller send and receive CAN messages. You do this with the `can_start` library routine.

```
can_start ();
```

Sending a message involves declaring a CAN message variable, initializing the message ID, setting the data, and calling the `can_send` library routine. For example:

```
struct can_message_struct t;
t.identifier = 0x0123;
t.c_data [0] = 0;
t.c_data [1] = 1;
can_send (&t);
```

Receiving a message is just as easy. You must declare a CAN message variable, bind the message ID to the

current task (`can_bind_obj` does that) and then call the `can_wait` library routine to wait for the message to be received. For example:

```
struct can_message_struct r;
can_bind_obj (0x0234);
can_wait (255, &r);
```

When `can_wait` returns, a message (with ID 0x0234) has been received.

Our engineers have created an application note for a simple CAN program using two MCB517 boards. The sample program transmits and receives a 1-byte value that displays on the board's Port 4 LEDs. This lets you see the application in action. You can use the dScope debugger to watch CAN

message variables and set breakpoints in the RTX51 tasks. Communication with MON51 makes *near real-time* debugging possible with the RTX51 kernel and the CAN controller.


RTX51 supports the Intel 82526/82527, the Philips 82C200, and Siemens 81C90/91 stand-alone CAN controllers. The Philips 8xC592 and 8xCE598 and Siemens C505C and C515C 8051-based microcontrollers with on-chip CAN controllers are supported as well.

Check the Keil Software web site for more information about the RTX family of Real-time Kernels.

<http://www.keil.com/rtos>

For more information about CAN and for web-based tutorials, see:

<http://www.keil.com/can>

Call us today for more information. 

## RTX51 CAN Support for Siemens C505C

Siemens makes two CAN-based 8051-compatible microcontrollers: the C505C and the C515C. The CAN libraries provided with the Keil RTX51 Real-time Kernel fully support both of these chips.

Support for the C515C is conspicuously present. The C505C, however, requires that you make two changes to the configuration file ICANCONF.A51 and add one statement in your initialization code.

The C505C, unlike the C515C, lets you locate the CAN controller base address at any 256-byte page in XDATA. This is done by setting the XPAGE SFR to the upper byte of the address.

First, define an SFR for XPAGE.

```
/* Define XPAGE SFR */
sfr XPAGE = 0x91;
```

Then, assign the upper byte of the base address to XPAGE. You may want to locate this at the same address as the C515C.

```
/* Set CAN base address */
/* 0xF7 is the same as C515C */
XPAGE = 0xF7;
```

Finally, you must modify the ICANCONF.A51 configuration file. The interrupt number (and possibly the base address) is different on the C505C. You must set the controller base address and interrupt number as shown below.

Once these things are done, you may use the RTX51 CAN Library as you would for a C515C application.

For more information about RTX51:

<http://www.keil.com/rtx51> 

```
CONTROLLER_BASE EQU 0F700H ; XDATA start-address of the
; CAN-Controller

; The following constant defines the interrupt-number (as defined by
; RTX51) for the CAN-Controller

;
; Should be: 17 for Siemens C515C
; 9 for Siemens C505C

USED_CAN_INT_NBR EQU 9
```

ICANCONF.H Configuration File

## Function Pointers

Function pointers are one of the many difficult features of the C programming language. Due to the requirements of the 8051 architecture, function pointers and reentrant functions have even greater challenges to surmount. This is primarily because of the way arguments are passed to functions.

Function arguments are typically passed on the stack using the push and pop instructions and are accessed using a base pointer. Since the 8051 has a size limited stack (from 256 bytes to as low as 64 bytes on some devices), function arguments must be passed using a different technique.

When Intel introduced the PL/M-51 compiler, they used a technique of storing arguments in fixed memory locations. When the linker was invoked, it built a call tree of the program, figured out which function arguments were mutually exclusive, and overlaid them. This was the beginning of the linker's **OVERLAY** directive.

Since PL/M-51 doesn't support function pointers, the issue of indirect function calls never came up. However, with C, problems abound. How does the linker know which memory to use for the indirect function's arguments? How do you add functions that are called indirectly into the call tree?

To help you better understand and use function pointers, we have a new application note on the web site. Application Note 129 discusses function pointers in C51. You may download the app. note from the web at:

<http://www.keil.com/appnotes>

There are several caveats you must be aware of if you use function pointers in your C51 programs.

## Argument List Limits

Arguments passed to functions called through a function pointer must all fit into registers. At most, three arguments can automatically be passed in registers. Refer to the C51 User's Manual to find out the algorithm for fitting arguments into registers.

Do not assume that just any three argument data types will fit.

Since C51 passes up to three arguments in registers, the memory space used for passing arguments is not an issue unless the function pointed to requires more arguments. If that is the case, you can merge the arguments into a structure and pass a pointer to the structure. If that is not acceptable, you may always use reentrant functions.

## Reentrant Functions

Keil C51 offers the **reentrant** keyword to declare functions that are reentrant. Reentrant functions expect arguments to be passed on a simulated stack. The stack is maintained in **IDATA** for small memory model, **PDATA** for compact memory model, or **XDATA** for large memory model. You must initialize the stack pointer in the startup code if you use reentrant functions. The following code listing shows the important definitions in **STARTUP.A51**.

```
-----  
; Reentrant Stack Initialization  
;  
; The following EQU statements define the stack pointer for reentrant  
; functions and initialized it:  
;  
; Stack Space for reentrant functions in the SMALL model.  
IBPSTACK EQU 0 ; set to 1 if small reentrant is used.  
IBPSTACKTOP EQU 0FFFH+1 ; set top of stack to highest location+1.  
;  
; Stack Space for reentrant functions in the LARGE model.  
XBPSTACK EQU 0 ; set to 1 if large reentrant is used.  
XBPSTACKTOP EQU 0FFFFH+1 ; set top of stack to highest location+1.  
;  
; Stack Space for reentrant functions in the COMPACT model.  
PPBSTACK EQU 0 ; set to 1 if compact reentrant is used.  
PPBSTACKTOP EQU 0FFFFH+1 ; set top of stack to highest location+1.  
-----
```

You must set the EQU to **1** for the memory model stack you use and set the address for the top of the stack.

The reentrant stack grows down as things are pushed onto the stack. The system stack grows up. For this reason, using **IDATA** for the reentrant stack is not always the best idea (even though it produces the smallest code). One way to conserve internal data memory (and allow for a larger system stack) is to place all reentrant functions in a separate memory model (like **large** or **compact**). The compact model generates less code than large model but it is more complex to configure.

To declare a reentrant function, use the **reentrant** keyword.

```
void func (  
    long arg1,  
    long arg2,  
    long arg3) reentrant  
{ ... }
```

To declare a large model reentrant function, use the **large** and **reentrant** keywords. For example:

```
void func (  
    long arg1,  
    long arg2,  
    long arg3) large reentrant  
{ ... }
```

## Pointers to Reentrant Functions

There is not much difference between declaring reentrant function pointers and non-reentrant function pointers. A reentrant function pointer merely requires that you use the **reentrant** keyword.

```
void (*rfunc_ptr) (  
    long a,  
    long b,  
    long c) reentrant = func;
```

When using pointers to reentrant functions, more code is generated because arguments must be pushed onto the simulated stack. However, no special linker controls are required and you need not learn anything about the **overlay** directive.

If you pass more than three arguments to a function that you call indirectly, reentrant functions may be what you need.



## Trade Shows

Keil Software exhibits at over 20 trade shows each year. From the Embedded Systems Conferences in San Jose and Chicago to Electronica to the Embedded Computing Shows in cities all over the USA. These shows give you an opportunity to meet us, see what new development tools we have, and allows us to help you with your embedded applications.

We offer FREE seminars on Controller Area Network (CAN) and Universal Serial Bus (USB) at many of the Embedded Computing Shows. These seminars give you the information you need to create applications with the Keil development tools and these popular communication protocols. They are also an excellent way for you to get a jump start on your next project.

Check our web site for the dates and times of trade shows in your area:

<http://www.keil.com/shows>

## Training Courses

In the United States, Keil Software and MicroConsult have partnered to bring you several new training courses.

- C166/C167 Hardware Design,
- C166/C167 Software Development,
- CAN Hands-on Workshop,
- Introduction to TriCore.

Training courses are taught each month in Dallas, TX, or San Jose, CA. Check our web site for the class schedule and itinerary.

<http://www.keil.com/class>

We continue to offer training courses for our C166 and C51 development tools and can custom tailor on-site training for your staff. Call our sales department at **800-348-8051** and register today.