

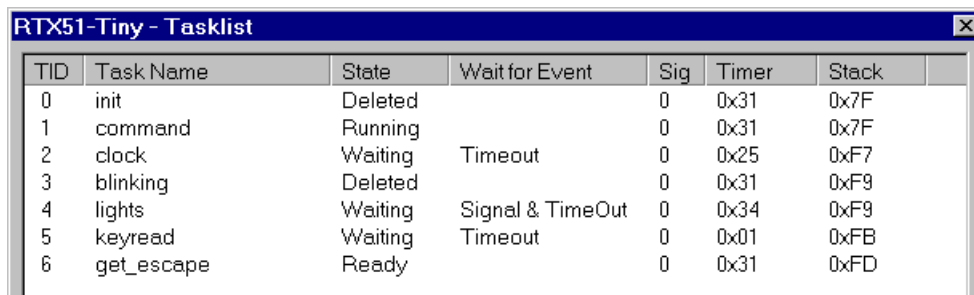
Implementing Display DLL's for User Defined Outputs

Application Note 144

Sept 21, 1999, Munich, Germany

by Peter Holzer, Keil Elektronik GmbH support.intl@keil.com ++49 89 456040-0

The μ Vision2 Debugger supports an interface to display DLL's. Display DLL's can be used to present task and other status information of real-time operating systems. However the capabilities are not limited to real-time operating systems, every kind of status information, i.e. the file table structure of smart card operating systems can be displayed.



TID	Task Name	State	Wait for Event	Sig	Timer	Stack
0	init	Deleted		0	0x31	0x7F
1	command	Running		0	0x31	0x7F
2	clock	Waiting	Timeout	0	0x25	0xF7
3	blinking	Deleted		0	0x31	0xF9
4	lights	Waiting	Signal & TimeOut	0	0x34	0xF9
5	keyread	Waiting	Timeout	0	0x01	0xFB
6	get_escape	Ready		0	0x31	0xFD

Example: RTX-51 Tiny Task List dialog

Select the User Display DLL within μ Vision2

The DLL Driver Name for external Display DLL's is stored in the file C:\KEIL\TOOLS.INI. A new driver is installed by adding the name and path to the C:\KEIL\TOOLS.INI file. Each CPU family has it's own section in the TOOLS.INI file.

Example for a TOOLS.INI file:

```
[UV2]
ORGANIZATION="Keil Elektronik GmbH"
:

[C166]
PATH="C:\Keil\C166"
BOOK0=HLP\RELEASE.TXT("Release Notes")
:
RTOS0=C:\CMX\BIN\UV2CMX.DLL ("CMX-166")
RTOS1=C:\RTXC\UV2RTXC.DLL ("RTXC-166")

[C51]
PATH="C:\Keil\C166"
RTOS0=C:\MYDLL\SMARTFILE.DLL ("SmartCard File System")
```

The display DLL's that should be used during system debugging can be selected in the **Options for Target - Target** page under **Operating System**. At the time the μ Vision2 debugger is started, the

display DLL is automatically loaded and initialized. The DLL dialog can be opened in the μ Vision2 menu under **Peripherals – DLL Defined Name**.

User Display DLL Interface Functions

A sample source code of a user display DLL can be found in the file **RTXTINY.CPP**. The following functions are used in context with the user display DLL interface.

Function:	Description:
BootDll	<p>Generic μVision2 interface function to the Display DLL. This function is called from the μVision2 debugger with the following function codes:</p> <p><u><i>nCode = 1:</i></u> init call to display DLL. <i>p1</i> is a data pointer to <i>struct bom</i> that contains interface functions and data of the μVision2 debugger. The display DLL may initialize the following pointers in the <i>struct bom</i>:</p> <p><i>pMrtx</i> address of menu array, see explanation below.</p> <p><i>RtxUpdate</i> address of update dialog function; called by μVision2 to update the information in all open dialogs.</p> <p><i>TaskRun</i> address of <i>_TaskRunning_</i> debug function. For a description of the usage refer to the <i>Getting Started and Creating Applications</i> User's Guide, Chapter 8 RTX Kernel Aware Debugging.</p> <p><u><i>nCode = 3:</i></u> 2nd init call to display DLL. <i>p1</i> is a data pointer to <i>struct dbgblk</i> that contains information about the target system and the debug environment.</p> <p><u><i>nCode = 4:</i></u> shut down call to display DLL. No further parameters passed. DLL must free all resources and close open dialogs.</p>
<i>pio</i>	<p>Is a pointer to <i>struct bom</i> that is passed with the <i>BootDLL</i> function call. The files BOM.H and COMTYP.H contain various definitions that are used within the following functions. The <i>struct bom</i> contains the addresses the following functions:</p>
<i>void</i> FetchItem	<p>(<i>UINT64 nAdr, TYP *tp, union v *pU</i>)</p> <p>Fetch CPU memory addressed by <i>nAdr</i> to <i>union v</i>. The data type (char, int, long, ...) is given by <i>TYP *tp</i>.</p>
<i>SYM *</i> FindPub	<p>(<i>char *name</i>)</p> <p>Returns the symbol information for a given public symbol <i>name</i>. A null pointer is returned if the search fails.</p>
<i>SYM *</i> PubSymByVal	<p>(<i>UINT64 nVal, DWORD nMask</i>)</p> <p>Returns the symbol information for a given value <i>nVal</i>. <i>nMask</i> denotes the symbol table to search for. For example, <i>nMask = F66_LOC</i> searches for function entries or assembler labels. <i>F66_VAR</i> searches for data symbols. A null pointer is returned if the search fails.</p>

Function:	Description:
<i>DWORD</i> ReadMem <i>DWORD</i> WriteMem	(<i>DWORD nAdr</i> , <i>DWORD nMany</i> , <i>BYTE *vp</i>) Read or Write to CPU memory at address <i>nAdr</i> . <i>nMany</i> is the number of bytes to transfer. <i>vp</i> is a pointer to the buffer. This functions returns a 0 if OK, otherwise the address is return where no memory access was possible.

Menu Structure

The μ Vision2 DDE interface allows you to implement several menu entries under the μ Vision2 peripheral menu include pop-up menus. The following lists the struct defintions that are used to define menu items. All definitions are in the file **BOM.H**.

```
#define DLGD struct DlgDat
struct DlgDat { // every dialog has it's own structure
    DWORD iOpen; // auto reopen dialog (pos := 'rc')
    HWND hw; // Hwnd of Dialog
    BOOL (CALLBACK *wp) (HWND hw, UINT msg, WPARAM wp, LPARAM lp);
    RECT rc; // Position rectangle
    void (*Update) (void); // Update dialog content
    void (*Kill) (DLGD *pM); // Kill dialog
    void *vp; // reserved for C++ Dialogs (Dlg *this)
};

#define DYM struct DynaM
struct DynaM { // Menu item data structure
    int nDelim; // Menu template delimiter: 1=normal entry, 2=popup entry,
    // -1=end of menu, -2=end of popup entry
    char *szText; // Menu item text
    void (*fp) (DYM *pM); // function to be activated on menu selection
    DWORD nID; // uv2 assigned ID_xxxx
    DWORD nDlgId; // Dialog ID
    DLGD *pDlg; // link to dialog attributes
};
```

Example for a user defined menu:

```
DLGD TaskDlg[] = { // must not use 'const' here !
//iOpen Hwnd Dlg Proc. Rect: -1 := default Update Fct Kill Fct
{ 0, NULL, NULL, { -1, -1, -1, -1, }, TaskUpdate, TaskKill },
};

DLGD Int0Dlg[] = { // must not use 'const' here !
//iOpen Hwnd Dlg Proc. Rect: -1 := default Update Fct Kill Fct
{ 0, NULL, NULL, { -1, -1, -1, -1, }, Int0Update, Int0Kill },
};

DLGD Int0Dlg[] = { // must not use 'const' here !
//iOpen Hwnd Dlg Proc. Rect: -1 := default Update Fct Kill Fct
{ 0, NULL, NULL, { -1, -1, -1, -1, }, Int0Update, Int0Kill },
};

DYM my_menu [] = {
//nDelim szText fp nID nDlgId pDlg
{ 1, "&Task Table" , TaskDisp, 0, IDD_TASK, &TaskDlg }, // Task Table display
{ 2, "&Interrupts" , NULL, 0, 0, NULL }, // Interrupt pop-up menu
{ 1, "Interpt. &0" , Int0Disp, 0, IDD_INT0, &Int0Dlg }, // Interupt-0 display
{ 1, "Interpt. &1" , Int1Disp, 0, IDD_INT1, &Int1Dlg }, // Interupt-0 display
{ -2, NULL , NULL, 0, 0, NULL }, // End of Port-Group
{ -1, NULL , NULL, 0, 0, NULL }, // End of Table
};
```

A sample dialog project for the RTX Tiny debug DLL's is available in source form as Microsoft Visual C project and can be freely modified. You can use this code license and royalty free within your application.