SIEMENS

SIMATIC

A Brief Introduction to Operating the M7 Debugger

Disclaimer of liability

We have checked the contents of this manual for agreement with the hardware and software described. Since de viations cannot be precluded entirely, we cannot guarantee full agreement. However, the data in this manual are reviewed regularly and any necessary corrections included in subsequent editions. Suggestions for improvement are welcomed.

SIMATIC® and SINEC® are registered trademarks of Siemens AG. Some of other designations used in these documents are also registered trademarks; the owner's rights may be violated if they are used by third parties for their own purposes.

Table of Contents

1 Introduction

This publication presents an introduction to using the Organon XDB M7 debugger, which is illustrated by means of two simple examples. The basic elements in running the debugger are explained in steps that you can easily repeat on your own.

Requirements

The example applications discussed in Sections 2 and 3 are described on the basis of the configuration listed below:

Hardware

An S7-300 rack with an S7 CPU 314 and an M7 FM 356-4 are used in the examples.

In addition to a power supply, your automation system should be equipped with the following modules as a minimum:

- in the case of an M7-300, an M7 FM behind an S7 CPU, or
- an M7 CPU, and
- a digital S7 output module

Connect your programmable logic controller to the programming device via the MPI. You will find a detailed description on the establishment of MPI connections in the STEP7 User Guide.

Software

The following software components should be installed on your programming device:

- SIMATIC STEP7 base system version 4.02 or higher
- M7-SYS system software version 2.00.14 or higher
- M7-ProC/C++ version 3.1
- BorlandC/C++ version 5.01

If older software versions are installed, you will not be able to use some of the functions described here.

Required knowledge

You should further be familiar with creating a project in STEP7 as well as writing C programs for the M7 using the ProC/C++ programming package and the Borland C/C++ compiler.

2 Example 1 Debugging a Simple Program

This example shows how to debug a simple M7 program containing a single task. It shows you how to

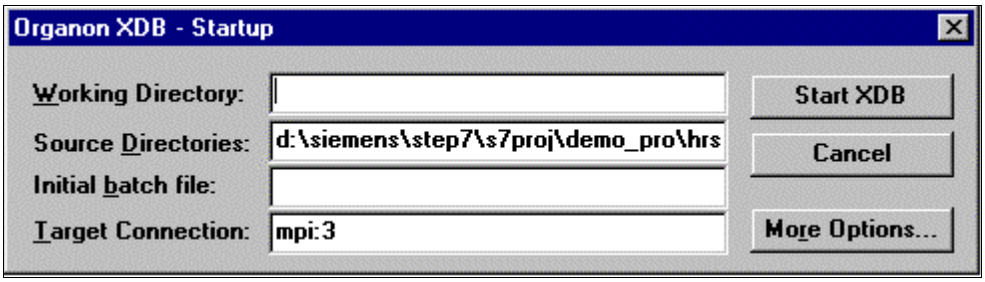
- call the debugger from the Borland IDE and establish the connection to the M7,
- load and start the program onto the M7,
- observe modifications of a variable,
- set and go to a breakpoint, and finally
- exit from the program and quit the debugger.

Compiling an executable program

First, create the example program introduced in Chapter 2 of the M7 programming manual /280/ in the manner it describes (it provides further details of the hardware configuration, among other things), but do not immediately download or start it on the M7 target system. The program does not have to be permanently stored on M7 target system in order for you to be able to use the debugger. The program is automatically loaded temporarily into the memory of the M7 target system for the duration of the debugging session and then removed again.

Calling the debugger

To start the debugger, choose in the Borland IDE **Tool ▸ STEP7 Debugger** from the menu. The following dialog box appears:



The entries in the **Working Directory, Source Directories** and **Initial batch file** fields can be ignored as far as our example is concerned. As you start the debugger from the Borland IDE, the requisite entries for this particular instance are performed automatically. Do not modify them.

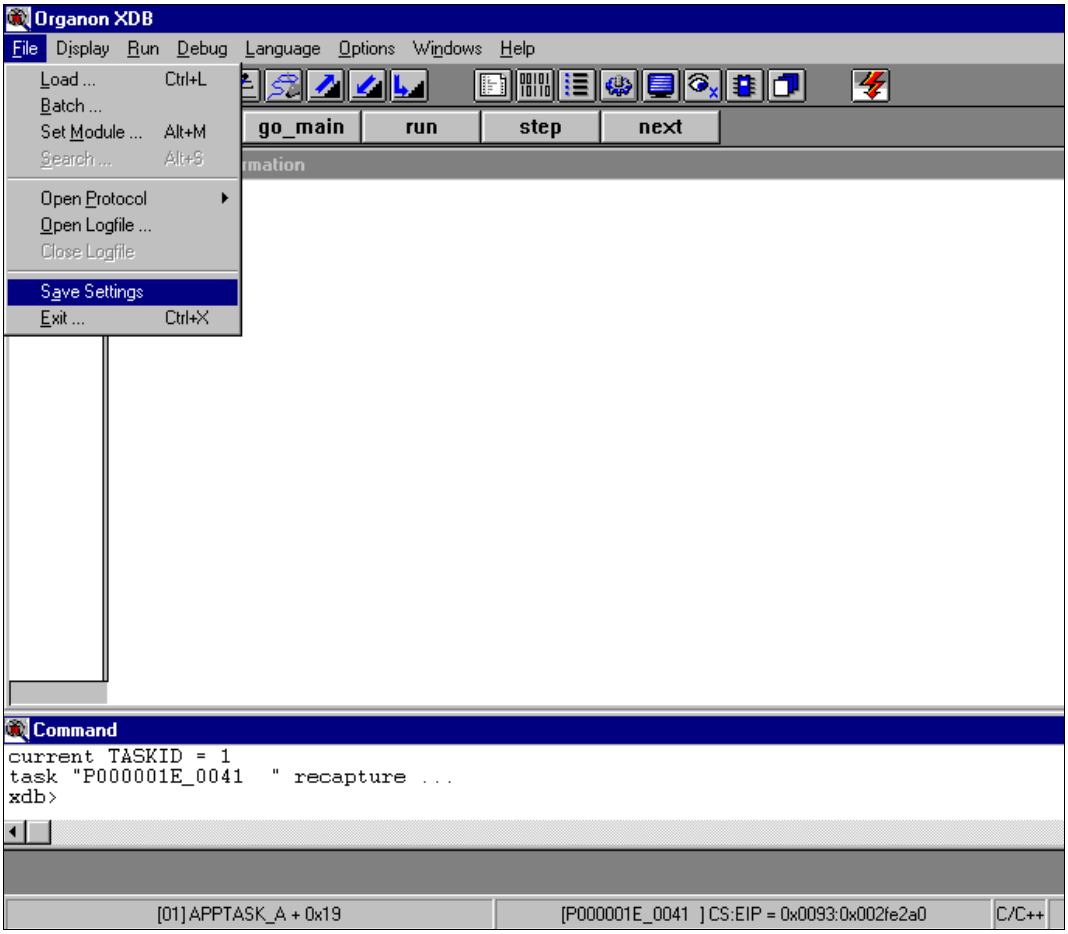
Enter the address of your M7 target system in the **Target Connection** field. You will find the address, for example, on the **General** tab of the **Object Properties** dialog box on the shortcut menu (right mouse button) of your M7 module in SIMATIC Manager (for example, MPI... 3).

Tip If you are working with an M7-400 station, you have to specify the address as

mpi:x,y, where x is the MPI address, and y the slot of the module. Y is calculated from the formula: 12 + 256 times slot number + 4096 times rack number. Enumeration of the rack and slot numbers starts at 0 so that y will normally be 12 (rack and slot numbers = 0)

Click the **Start XDB** button to start the debugger.

Once the debugger has been started successfully, its main window appears. Size the **Source** and **Command** windows and choose **File ▸ Save Settings** from the menu.

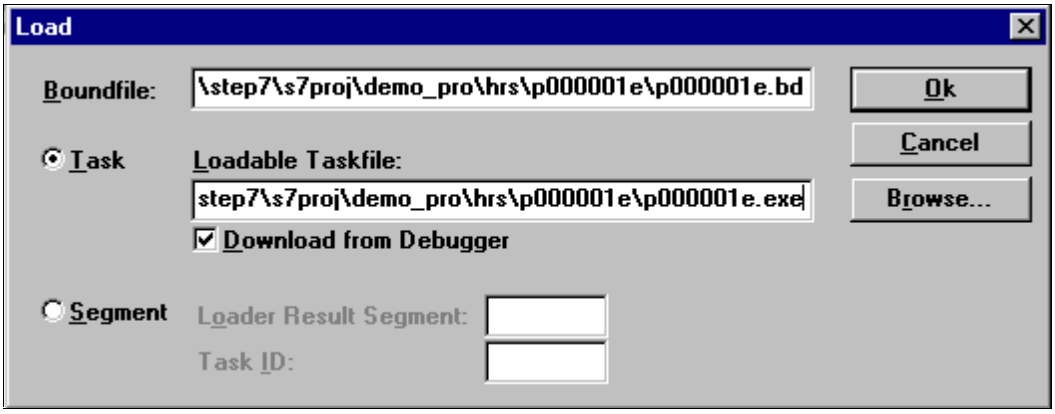


This saves the size and arrangement of the windows for subsequent sessions.

Loading and starting the program

After the debugger has been started and the MPI connection has been established between the programming device and the M7, load the program onto the debugger. Load the executable file on the M7 and start it on the M7.

To do that, first choose in the debugger **File ▸ Load** (shortcut **Ctrl + L**) from the menu. By default, the bound and task files of the last task that was processed are entered in the **Load** dialog box.



If you wish to debug a different task, enter in the **Boundfile** field the path and name of the **.bd** file of your program. Select **Task** and enter in the **Loadable Taskfile** field the path and the name of the executable **.exe** file of your program. Depending on which of the two fields you have just selected, you can search for the corresponding file on your hard disk using **Browse**, should you wish. Further, check the **Download from Debugger** option so that the M7 debugger automatically loads the program onto the M7.

Click **OK**. The task is now loaded onto the M7 and logged on at the operating system (in other words, created and catalogued). The program still cannot be seen on the screen.

The M7 debugger assigns the internal task ID 1. Click the **set_task** button and enter task ID 1. By doing this, you select the loaded program as the current task in the M7 debugger and start it (you will find the assignment of the task ID by choosing **Display ▸ Tasks** from the menu). However, the M7 debugger stops execution immediately prior to executing the first instruction.

If you click the **go_main** button, the startup code of the program is executed and the source code is visible in the **Source** window. The debugger stands at the function entry code of the **main** function.

Tip The tabulator width of the source window is set to 8 by default. You can modify it by setting the **XDBTABLEN** environment variable in your autoexec.bat file. The entry

```
set XDBTABLEN=4
```

sets the tabulator width to 4, for example.

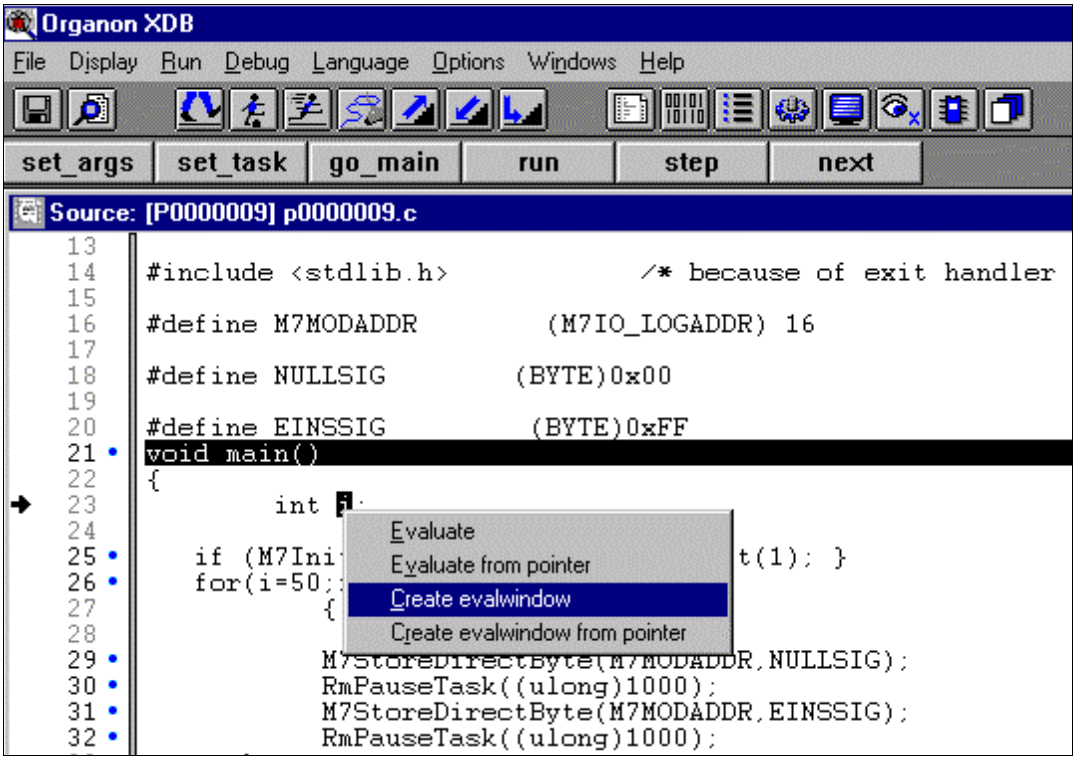
Running the program

Let the debugger process the program line by line by repeatedly clicking the **step** button.

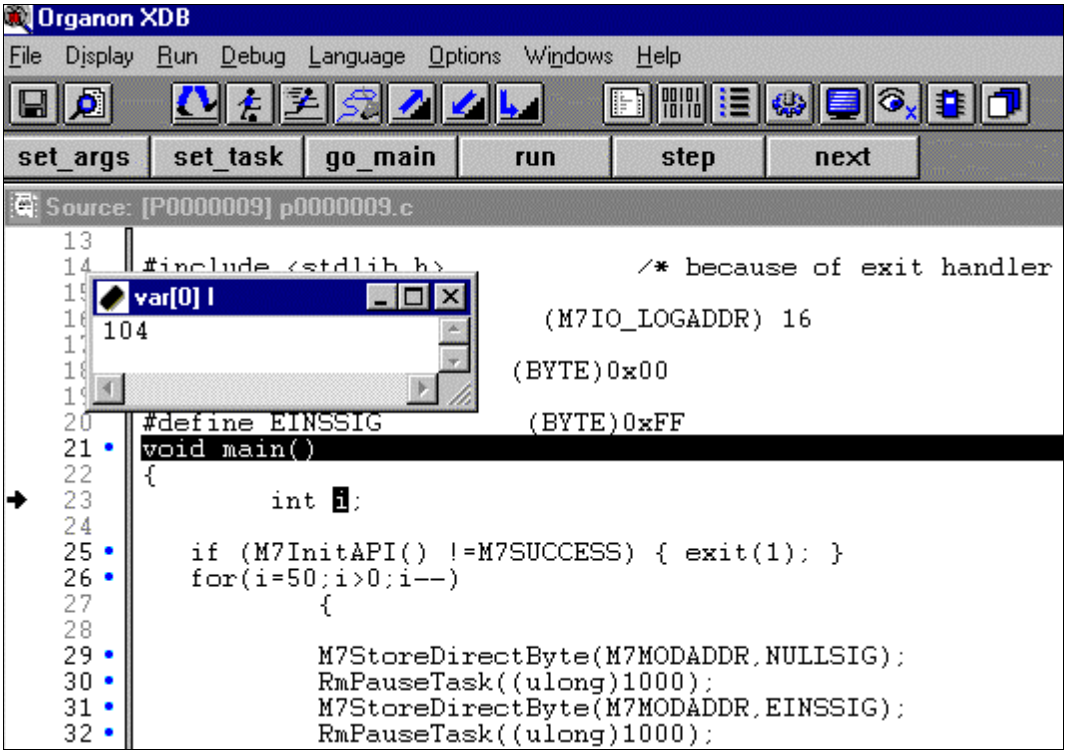
As soon as the debugger gets to the end of the program, it stops processing the task. If you want to process the task again, select the required task number again by clicking the **set_task** button and going to the main program with the **go_main** button.

Observing a variable

Let us assume that you want to observe variable “i“ in order to follow how it progresses while the program is being executed. The simplest way of doing this is to view the variable in an **evalwindow**. To do so, select the variable and press the right mouse button.

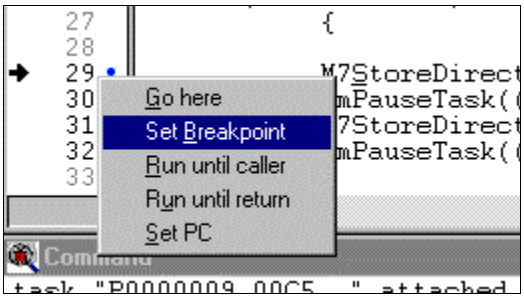


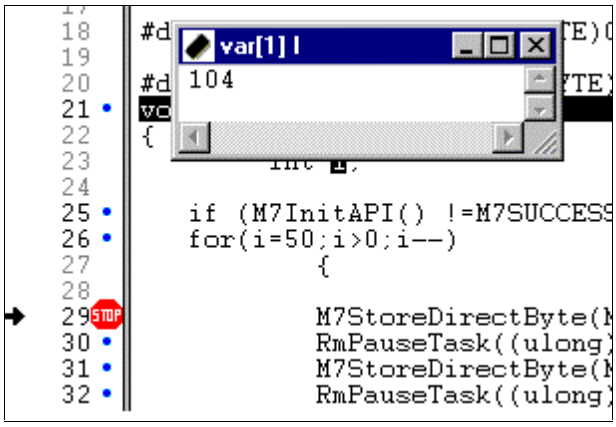
Select **Create Evalwindow** on the shortcut menu that is displayed. A window, in which the current value of variable “i“ is displayed, is opened.



Setting a breakpoint

Set a breakpoint on line 29 (the first line of the loop that is executed i times). To do this, move the mouse pointer to line number 29 in the left section of the **Source** window, press the right mouse button and choose **Set Breakpoint** from the shortcut menu that is displayed. You can similarly set (or clear) a breakpoint by double-clicking the line number.

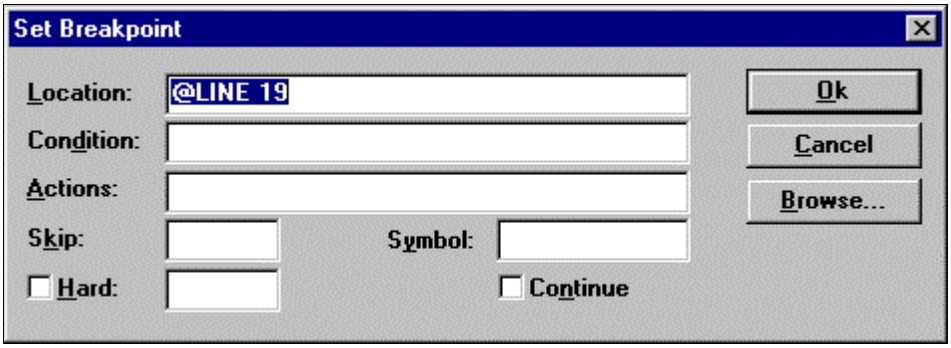




Tip Breakpoints can be set only on lines which are marked with a blue dot following the line number.

Breakpoints are displayed in the program by means of a red stop sign following the line number.

As an alternative you have the option of linking specific conditions and actions to the reaching of a breakpoint. To do this, press **Ctrl + B**. In the window that opens enter the line number for the breakpoint and the conditions and actions, and confirm by clicking **OK**.



Then click **run**. The program is executed until it reaches the breakpoint.

In the **Evalwindow,** the value of the variable has changed to 50 (a change is indicated by means of red writing). By repeatedly clicking the **run** button the program is executed each time until it reaches the breakpoint, and you can follow in the **Evalwindow** how the value of the variable is decremented by one each time.

Exiting from the debugger

After the program has finished running - in other words, when the debugger has reached the **exit** instruction) the debugger terminates the task and the source code disappears from the source window. Choose the **File ▸ exit** menu command to exit from the debugging session in a normal manner. The user interface of Organon XDB is closed, and you are back in the Borland IDE.

3 Example 2 Debugging Several Tasks

This example will show you how to debug several tasks that are created, executed and deleted within an **.exe** file.

Multi-tasking function

In version 2.00.14 or higher of M7-SYS the Organon XDB STEP7 debugger is capable of multitasking. This means that you can simultaneously process several tasks associated with an **.exe** file in a single debugging session.

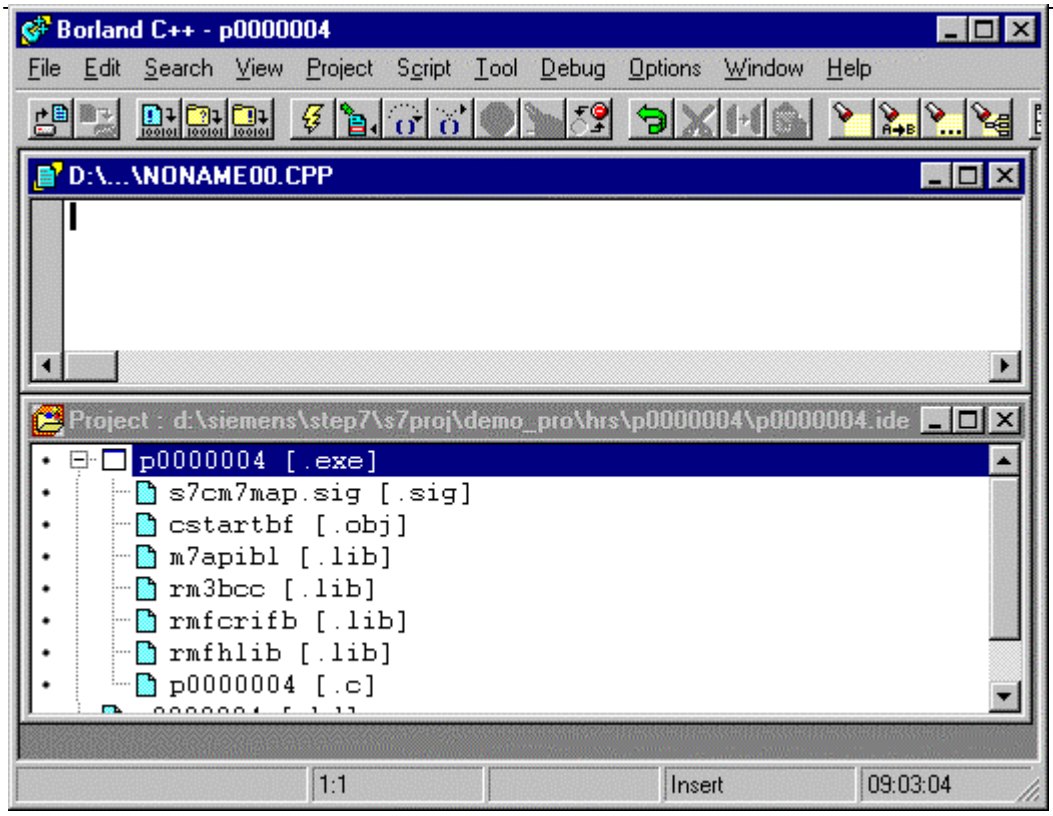
Tip If the “change from compatible to asynchronous debug mode failed“ message appears in the **Command** window after the debugger has been started, you have an old version of M7-SYS installed on your M7. In this case you cannot debug several tasks simultaneously with the debugger.

Copying an executable program with several tasks

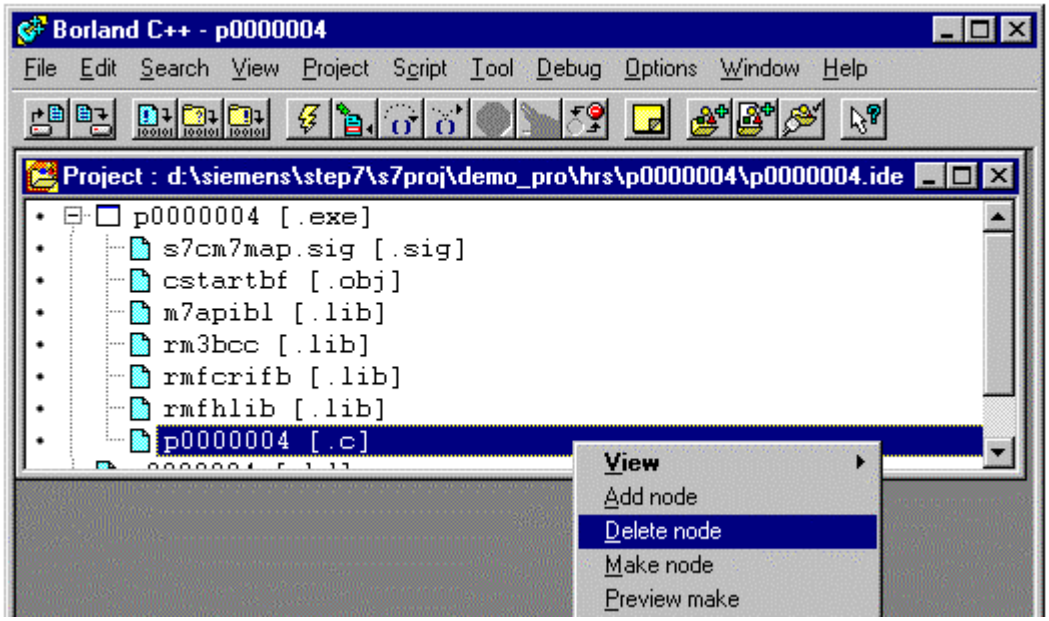
In this example we will use the source code of the “startup“ program, which is included in your M7-SYS software under m7sys4.00\examples\m7api\anlauf. For this purpose you should first copy the program to the corresponding directory in your project:

- Open the project you created.
- Select the M7 program in SIMATIC Manager.
- Choose **Paste ▸ M7 Software ▸ C Program** from the menu.
- Change the name of the new program to “startup“.
- Open the Borland IDE by double-clicking “startup“.

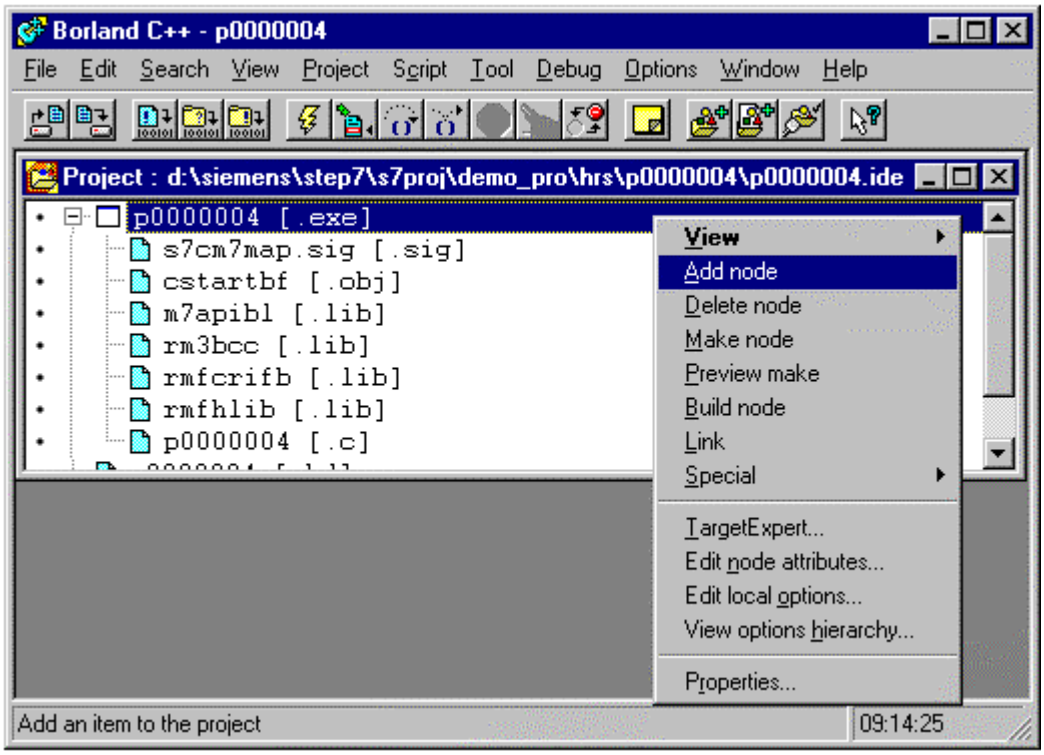
The Borland IDE is started, and the project window and the window for the source code having the extension **.cpp** are displayed.



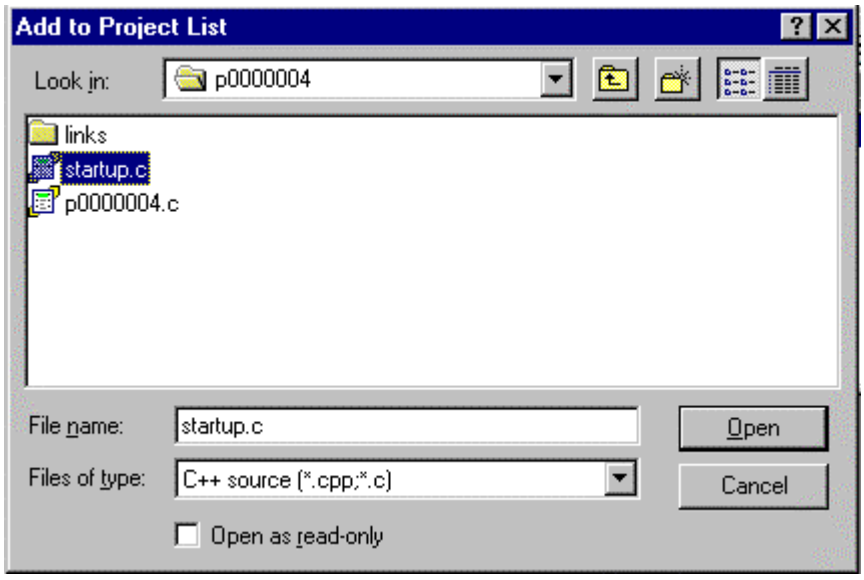
- Close the source code window and select the file with the **[.c]** extension in the project window.
- Press the right mouse button and chose **Delete Node** from the menu, and confirm by clicking **OK**.



- Open Explorer as the next step.
- Copy in Explorer the file **Siemens\Step7\M7Sys\examples\M7api\anlauf** (file type: C source file) to **Siemens\Step7\s7proj\<S7-ProjectName>\hrs\<Name of Source Folder>**. You can take this path from the title bar of the project window in the Borland IDE, for example.
- Change back to the Borland IDE.
- Select the source folder ([.exe] extension) in the project window.
- Press the right mouse button and choose **Add Node** from the menu.
- Rename the file from “anlauf.c” to “startup.c”.



- In the dialog box that appears double-click **startup** to select the program.



If the program called **startup** is not displayed in the corresponding folder, you have presumably not specified the correct destination path in the **copy** step. In this case repeat the **copy** step, specifying the correct path.

The new node, **startup.c**, is now displayed in the project window.

- Double-click the **startup.c** node.

The source code of the “startup“ program appears.

- Click the **Build Project** button to **recompile** a**nd link** the program.

- Acknowledge the **Success** message by pressing **OK**.

You have now added the program to your project and translated it.

Calling the debugger and starting the program

Call the debugger as described in the previous example:

- Choose **Tool ▸ Step7 Debugger** from the menu and click the **Start XDB** button.
- Load the program using **Ctrl + L**, confirm by clicking **OK**.
- Click the **set_task** button.
- Enter 1 and confirm by pressing Return.
- Click the **go_main** button.

The debugger goes to the first line of the main program.

Setting breakpoints

You already learned how to set breakpoints in the first example:

- Set breakpoints on lines 158, 191, 206, 209, 239, 243 and 245.

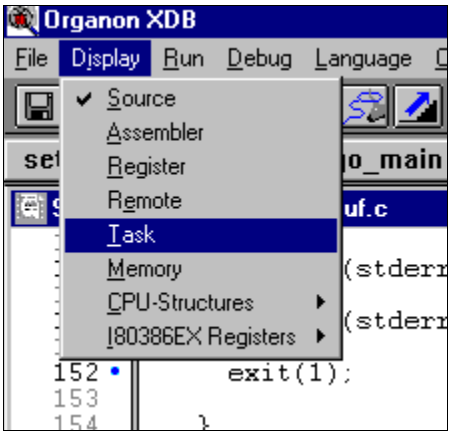
- To continue the program, press the **run** button.

The debugger goes to line 159.

Changing the active task

Open the task window to observe the operating states of the different tasks. You can also switch from one task to another starting from this window.

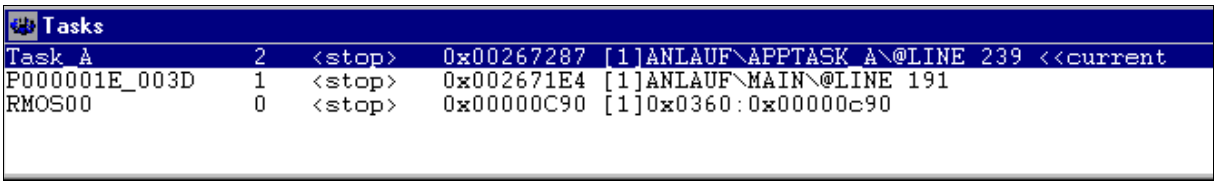
- To do so, choose **Display ▸ Task** from the menu.



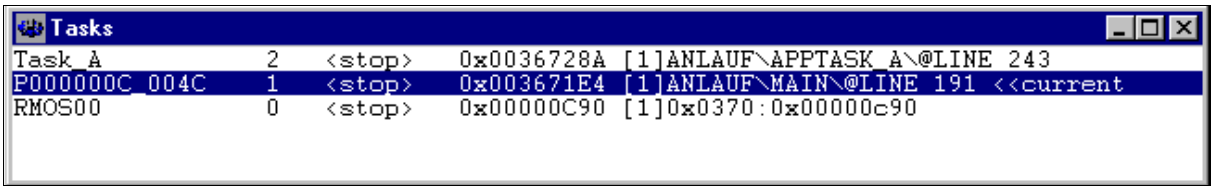
In the task window that opens two tasks known to the debugger are displayed - the main task and the RMOS 00 for the operating system itself, the current task being labeled "**<<current"**.

- Press the **run** button.

The debugger goes to line 191 in which the priority of the main task is reduced, in other words task_A becomes active, it is now displayed as the third task in the **Tasks** window and labeled as the current task. The debugger goes immediately to line 239, the first line of task_A.



You can now toggle between the tasks in the **Tasks** window by double-clicking on the task concerned and can thus debug both tasks:



- Double-click the main task.

  ➔The debugger goes back to line 191.

- Click the **run** button.

  ➔The debugger goes to the next breakpoint in the main task.

- Double-click task_A in the **Tasks** window.

  →The debugger goes to the last line processed in task_A.

- Click the **run** button three times to run task_A in three steps.

  →The debugger processes task_A step by step and exits from it. This is then displayed in the **Tasks** window by means of the <**exit>** operating state.

- Double-click the main task in the **Tasks** window.

  →The debugger goes back to line 206.

- Click the **run** button.

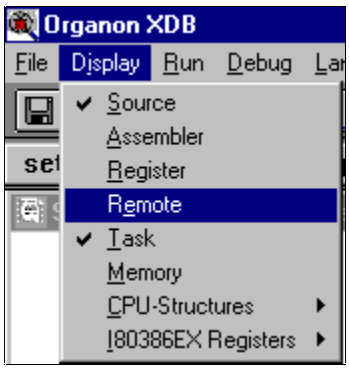
  →Task_A is deleted, meaning that it is no longer displayed in the **Tasks** window.

Exiting from the debugger

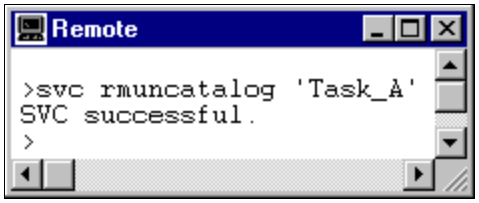
To exit from the debugging session, choose **File ▸ Exit** from the menu. The user interface of Organon XDB is closed, and you are back in the Borland IDE..

Tip If task_A is not terminated properly, and the **RmDeleteTask** instruction cannot be executed in the main function, the entry for task_A remains in the resources catalog. In this case the program is quit the next time it runs following **RmCreateTask** being called since task_A cannot be entered a second time in the catalog. Task_A has to be removed manually from the catalog.

To do this, choose **Display ▸ Remote** from the menu.



In the Remote window, enter the command **svc rmuncatalog ´Task_A´**.



The message **SVC successful** appears.

4 Bibliography

Excerpt:

/80/ Manual: M7-300 Programmable Controller, Hardware and Installation

/231/ User Manual: Standard Software for S7and M7, STEP 7

/280/ Programming Manual: System Software for M7-300/400, Program Design

/281/ Reference Manual: System Software for M7-300and M7-400, System and Standard Functions

/282/ User Manual: System Software for M7-300and M7-400, Installation and Operation

/290/ User Manual: ProC/C++ for M7-300 and M7-400, Writing C Programs

