SIEMENS

SIMATIC NET
Industrial Communication

MOBIC T8 for Windows CE 3.0

Programming Instructions

C79000-G8976-C191-01

Release 12/2003

Classification of Safety-Related Notices

This manual contains notices which you should observe to ensure your own personal safety, as well as to protect the product and connected equipment. These notices are highlighted in the manual by a warning triangle and are marked as follows according to the level of danger:

Danger

indicates that death, severe personal injury or substantial property damage **will** result if proper precautions are not taken.

Warning

indicates that death, severe personal injury or substantial property damage **can** result if proper precautions are not taken.

Caution

indicates that minor personal injury or property damage can result if proper precautions are not taken.

Caution

indicates that property damage can result if proper precautions are not taken.

Notice

highlights important information on the product, using the product, or part of the documentation that is of particular importance and that may have detrimental effects if ignored.

Note

highlights important information on the product, using the product, or part of the documentation that is of particular importance and that will be of benefit to the user.

Trademarks

Safety Instructions Regarding your Product:

Before you use the product described here, read the safety instructions below thoroughly.

Qualified Personnel

Only **qualified personnel** should be allowed to install and work on this equipment. Qualified persons are defined as persons who are authorized to commission, to ground, and to tag circuits, equipment, and systems in accordance with established safety practices and standards.

Correct Usage of Hardware Products

Note the following:

Warning

This device and its components may only be used for the applications described in the catalog or the technical description, and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens.

This product can only function correctly and safely if it is transported, stored, set up, and installed correctly, and operated and maintained as recommended.

Before you use the supplied sample programs or programs you have written yourself, make certain that no injury to persons nor damage to equipment can result in your plant or process.

EU Directive: Do not start up until you have established that the machine on which you intend to run this component complies with the directive 89/392/EEC.

Correct Usage of Software Products

Note the following:

Warning

This software may only be used for the applications described in the catalog or the technical description, and only in connection with software products, devices, or components from other manufacturers which have been approved or recommended by Siemens.

Before you use the supplied sample programs or programs you have written yourself, make certain that no injury to persons nor damage to equipment can result in your plant or process.

Preface 1

Purpose of the Manual

This manual is intended to help solution partners in the development of new applications for the MOBIC. It explains the specific functions of the MOBIC that allow special features of this device to be used.

Aims

With this manual, you should be able to use the programming interface described here.

Validity of the Manual

This manual applies to the following software versions:

- WINDOWS CE 3.0
- MOBIC SDK V3.0.1

Other Documentation

We assume that you are familiar with the MOBIC manual.

Finding Information

To help you find information more quickly, the manual includes not only a table of contents but also the following sections in the Appendix:

- Glossary
- References
- Index

Internet Address

In our Web site, you will find wide-ranging information about the product, customer support, and tips and tricks to help you make the best use of your MOBIC.

www.siemens.com/mobic

Audience

This documentation is intended for developers of application software.

Prerequisites

The user should have experience of programming with the following:

- Visual C++ or Visual Basic.

MOBIC Toolkit Training

To familiarize you with the MOBIC, we also offer a training course.

The MOBIC Toolkit package includes the following:

- MS-Embedded Visual Tools 3.0
- MOBIC Programming Instructions
- 1 day of training
- 8 hours of support

Development Environment 1

1.1 Hardware Requirements

Windows CE applications are developed on a PC. According to Microsoft, the minimum system requirements for Windows CE development are as follows:

Minimum Requirements

- PC with Pentium 90 MHz,
- 32 MB RAM
- 610 MB hard disk

Ideal Hardware Configuration

You achieve better results, however, with

- PC with at least a Pentium II 300 MHz
- 128 MB RAM
- 10 GB hard disk

1.2 Additional Programs Required for Software Development

Operating Systems

One of the following operating systems can be used:

- Windows 98
- Windows NT (Windows CE emulation possible)
- Windows 2000 (Windows CE emulation possible)

We recommend that you use Windows NT or Windows 2000.

Required Tools

- eMbedded Visual Tools 3.0

  To develop applications for the MOBIC, you require “eMbedded Visual Tools 3.0” from Microsoft. This software package contains the compilers to allow you to develop applications for Windows CE with Visual C++ and Visual Basic.

  You can download eMbedded Visual Tools 3.0 free from the Internet:

  www.microsoft.com/downloads

  or

  eMbedded Visual Tools 3.0 is included in the MOBIC Toolkit
  (see Preface “MOBIC Toolkit Training”).

1.3 Installation of the Development Environment

If you encounter problems during installation of the following programs or when creating applications, it is advisable to reinstall Windows. This restores the system to a defined status. The order in which you install the programs is extremely important.

Installation with eMbedded Visual Tools 3.0

1. Install the operating system.
2. Install ActiveSync.
3. Install “eMbedded Visual Tools 3.0” .
4. Install MOBIC SDK.

   The MOBIC SDK is on the MOBIC CD in the “\SDK” folder.

   There is an SDK
   - for Visual C++ with the name “MOBIC_vc_3_0_1.exe” and
   - for Visual Basic with the name “MOBIC_vb_3_0_1.exe”.

   Install the SDK for the programming language you intend to use. If you wish, you can also install both SDKs. Installation is started by clicking the file.

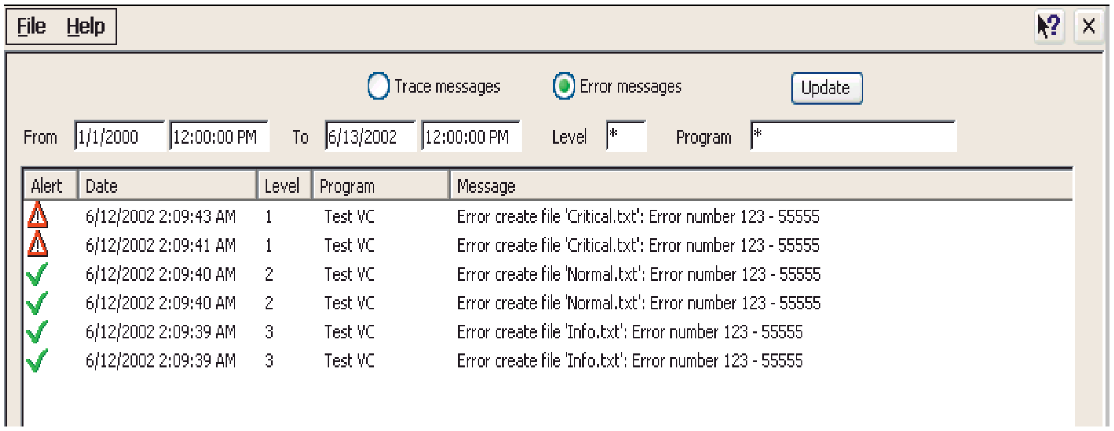
Tools for Software Development 2

2.1 Troubleshooting

For troubleshooting during development and later use in the field, MOBIC provides certain integrated tools. These allow applications to write important information to an error archive using the **Error Module** and to a trace archive using the **Trace Module**.

These modules are configured by the **MOBIC Configurator** integrated on the MOBIC. These archives are analyzed by the **LogView** that is also integrated on the MOBIC.

You can use these troubleshooting tools in conjunction with both Visual Basic and Visual C++. These tools are intended mainly for solution partners as support during the development of applications and for later maintenance and troubleshooting in the field if customers have problems.



From 1/1/2000 12:00:00 PM To 6/13/2002 12:00:00 PM Level * Program *

Alert	Date	Level	Program	Message
⚠	6/12/2002 2:09:43 AM	1	Test VC	Error create file 'Critical.txt': Error number 123 - 55555
⚠	6/12/2002 2:09:41 AM	1	Test VC	Error create file 'Critical.txt': Error number 123 - 55555
✓	6/12/2002 2:09:40 AM	2	Test VC	Error create file 'Normal.txt': Error number 123 - 55555
✓	6/12/2002 2:09:40 AM	2	Test VC	Error create file 'Normal.txt': Error number 123 - 55555
✓	6/12/2002 2:09:39 AM	3	Test VC	Error create file 'Info.txt': Error number 123 - 55555
✓	6/12/2002 2:09:39 AM	3	Test VC	Error create file 'Info.txt': Error number 123 - 55555

Figure 2-1 Error Archive

2.2 Error Module Tool

The Error Module informs the operator of errors and archives all relevant events in an error archive.

Error Levels

During development of your applications, you decide which errors will be assigned to which error level.

There are three different error levels:

- Critical Error
- Normal Error
- User Information

Error Output

To use this tool, you must use the error output function in your program code (see Section 3.2).

Please note that only errors should be signaled here. If you want to output information, for example, about the status of your application, use the Trace Module.

Error IDs

To increase the number of entries possible in the error archive, error IDs are stored instead of the complete error texts. The actual error text and its error ID are stored in a separate error text file.

The error output functions, their integration in program code, and creating error text files are described in greater detail in Section 3.2.

Settings

Using the MOBIC Configurator, you can customize the way in which the Error Module works.

For more detailed information, refer to the MOBIC manual.

2.3 Trace Module Tool

Definition

Traces are text outputs that you can use for troubleshooting and to obtain information during the development of your applications and during later use in the field.

Trace Levels

There are three different levels for traces.

- **Important trace** is intended for extremely important output which should be kept as short as possible.
- **Extended trace** is intended for additional information.
- **Detailed trace** is intended for extremely detailed output.

Settings

You can customize the properties of the Trace Module using the MOBIC Configurator.

For more detailed information, refer to the MOBIC manual.

2.4 Analyzing Error and Trace Outputs

LogView

The error and trace outputs created by the Error and Trace modules are analyzed using the **LogView** integrated on the MOBIC. This displays the error archive and the trace archive.

For more detailed information, refer to the MOBIC manual.

3 Programming Interface

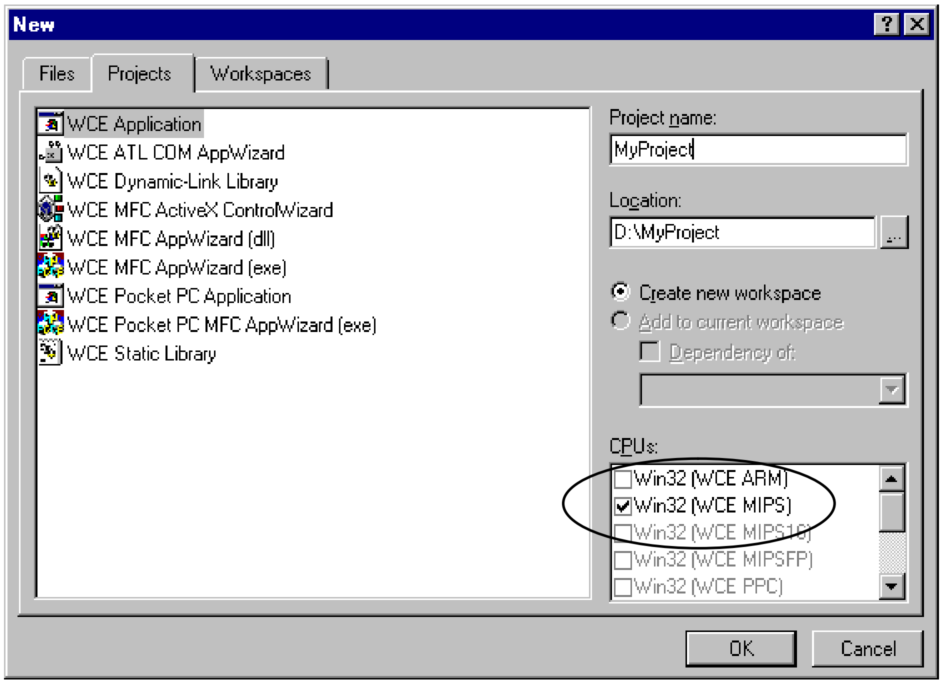
3.1 Creating the Project

You can develop applications for Windows CE both with Visual C++ and Visual Basic.

1. Install the development environment including the MOBIC SDK as described in Chapter 2.
2. Create a connection using ActiveSync (serial or Ethernet).
3. Create a new Windows CE project.

   Visual C++

   With Visual C++, the names of projects start with the letters “WCE ...” standing for Windows CE. “Win32 (WCE MIPS)” must be selected as the CPU.



Project name: MyProject

Location: D:\MyProject

[✓ Create new workspace] Add to current workspace

CPUs:
- ☐ Win32 (WCE ARM)
- ☑ Win32 (WCE MIPS)
- ☐ Win32 (WCE MIPS16)
- ☐ Win32 (WCE MIPSFP)
- ☐ Win32 (WCE PPC)

Figure 3-1

Select “MOBIC” as the active WCE configuration (see Figure 3-2).

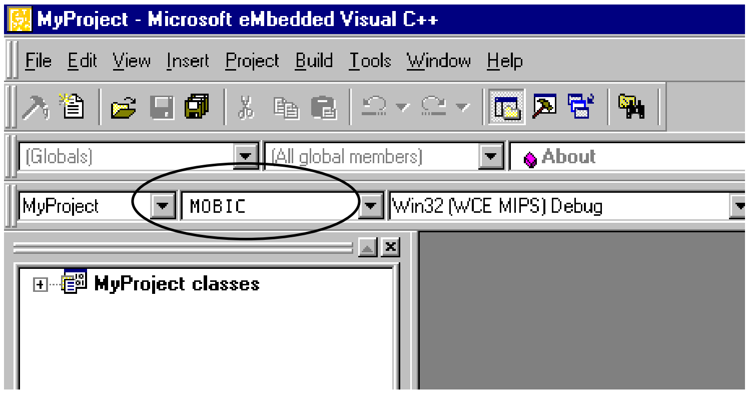

MyProject - Microsoft eMbedded Visual C++
File Edit View Insert Project Build Tools Window Help
(Globals)
(All global members)
About
MyProject
MOBIC
Win32 (WCE MIPS) Debug
MyProject classes

Figure 3-2

Using the “Tools” -> “Configure Platform Manager..” menu item, set the previously selected ActiveSync connection type (see Figure 3-3).

Select the “Default” entry under MOBIC in the “Windows CE Platform Manager Configuration” dialog and then click “Properties”.

Select “Microsoft ActiveSync” in the “Device Properties” dialog .

Confirm your selection with OK.

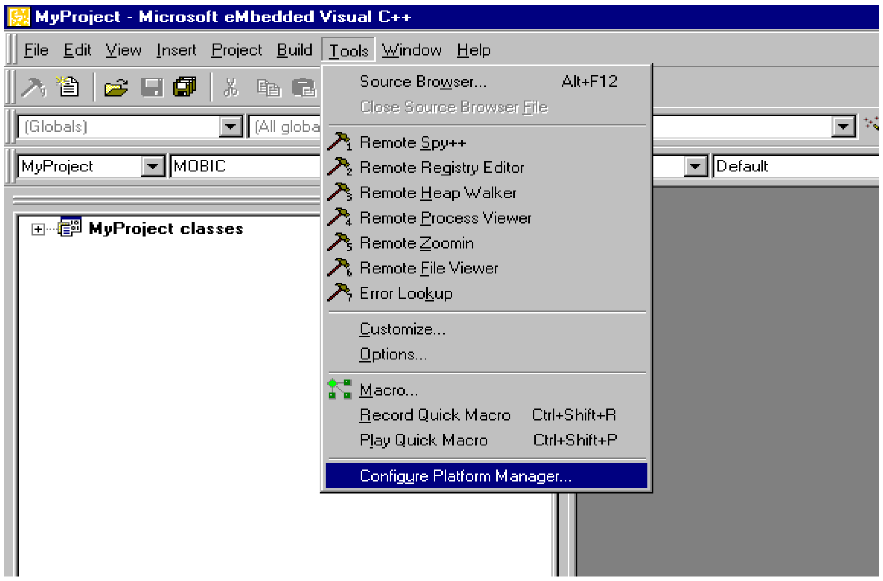

MyProject - Microsoft eMbedded Visual C++
File Edit View Insert Project Build Tools Window Help
(Globals)
(All globa
MyProject
MOBIC
Default
MyProject classes
Source Browser... Alt+F12
Close Source Browser File
Remote Spy++
Remote Registry Editor
Remote Heap Walker
Remote Process Viewer
Remote Zoomin
Remote File Viewer
Error Lookup
Customize...
Options...
Macro...
Record Quick Macro Ctrl+Shift+R
Play Quick Macro Ctrl+Shift+P
Configure Platform Manager...

Figure 3-3

Visual Basic

With Visual Basic, the names of projects start with “Windows CE ...” to indicate Windows CE.

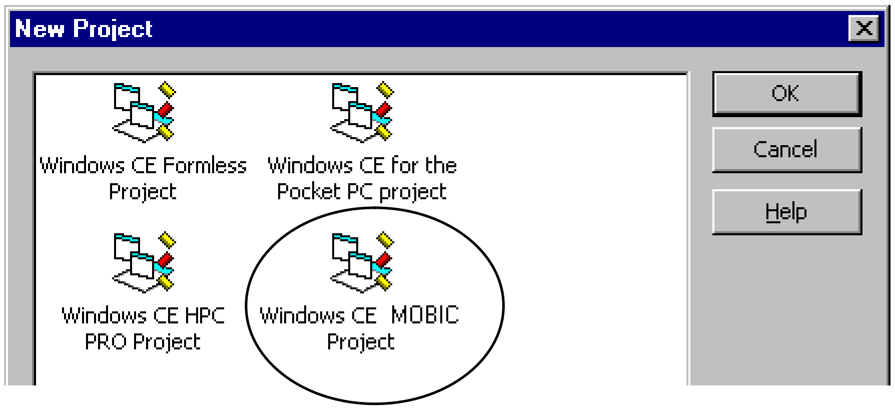

New Project
Windows CE Formless Project
Windows CE for the Pocket PC project
Windows CE HPC PRO Project
Windows CE MOBIC Project
OK
Cancel
Help

Figure 3-4

Using the “Tools” -> “Remote Tools”-> “Configure Platform Manager..” menu command, set the ActiveSync connection type (see Figure 3-5).

Select the “Default” entry under MOBIC in the “Windows CE Platform Manager Configuration” dialog and then click “Properties”.

Select “Microsoft ActiveSync” in the “Device Properties” dialog .

Confirm your selection with OK.

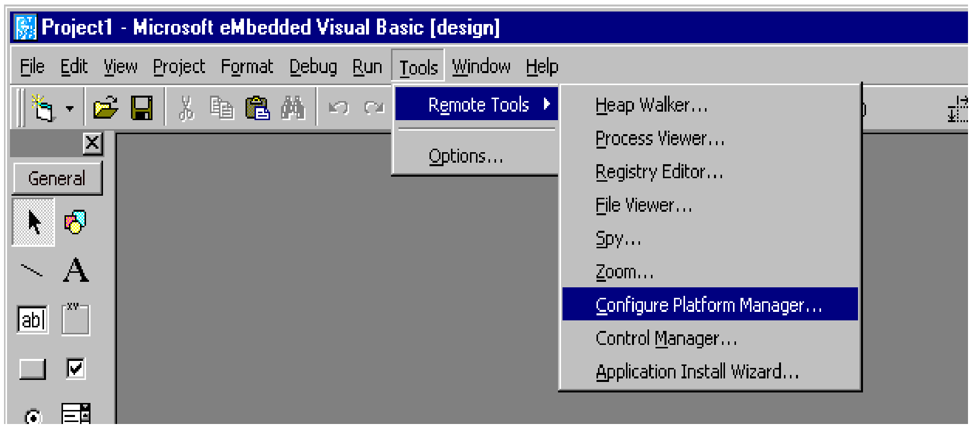

Project1 - Microsoft eMbedded Visual Basic [design]
File Edit View Project Format Debug Run Tools Window Help
Remote Tools
Options...
Heap Walker...
Process Viewer...
Registry Editor...
File Viewer...
Spy...
Zoom...
Configure Platform Manager...
Control Manager...
Application Install Wizard...
General

Figure 3-5

3.2 Error Module

The functionality of the error module is in the DLL “**ICError.dll**”. If you want to use these functions, you must first make them known in the development environment.

Visual C++:

In Visual C++, you add the relevant header file and library to your project.

The header file is included in your program code with the “**#include <icerror.h>**” statement. This contains all the necessary declarations.

The library is added with Project->Add to Project->Files.

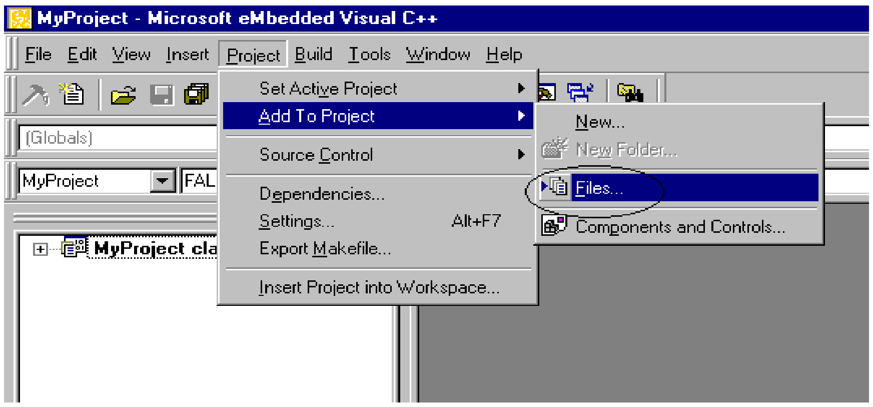


Figure 3-6

The required library “**ICError.lib**” is located in **\Windows CE Tools\wce212\MOBIC\Lib\Mips**. To display the library files, you must select “**Library Files (.lib)**” as the file type.

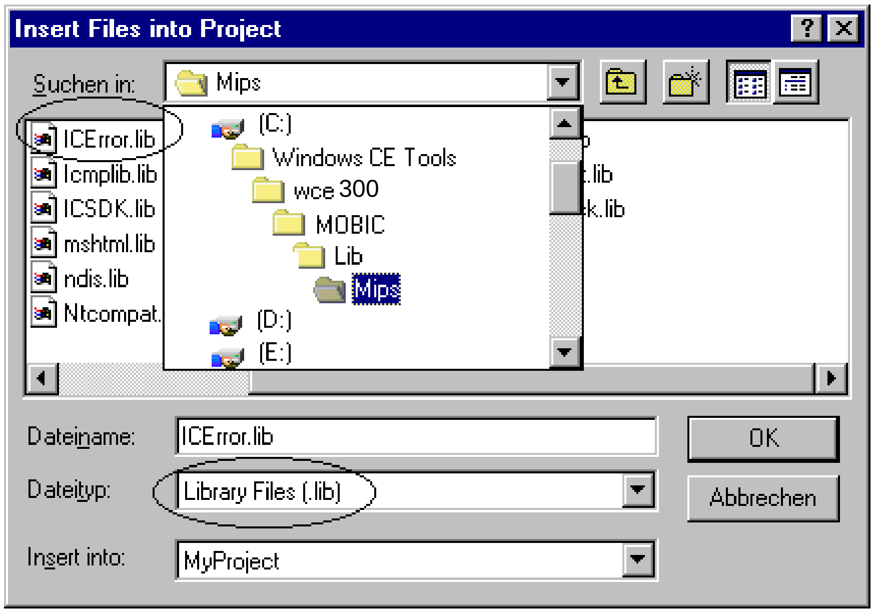


Figure 3-7

Note

When you installed the MOBIC SDK (see Section 2.3), you specified the folder in which the SDK is stored. The default folder proposed during installation is “Windows CE Tools”. If you selected a different folder, you will find the library file there.

Visual Basic:

In Visual Basic, you do not need to add the header file or library to your project. You must declare each function of the Error Module used. To do this, you must add a line to the declaration section of the program.

If, for example, you want to use the **ICErrorExit** function in your program, enter the following line in the “**declarations**” section:

```
Declare Function ICErrorExit Lib “ICError.dll” () As Boolean
```

3.2.1 Functions

ICErrorInit(...)

The *ICErrorInit* function initializes the Error Module. It must be called once before the first error output. We recommend that you call it right at the start of your program.

Visual C++ declaration (contained in the header file):

```
BOOL ICErrorInit(LPCTSTR szAppName)
```

Visual Basic Declaration:

```
Declare Function ICErrorInit Lib “ICError.dll” _
            (ByVal szAppName As String) As Boolean
```

Call parameters:

`szAppName`
With this parameter, you can transfer the name of your program as a string. The application name must not exceed 32 characters in length.

Return value:

`TRUE,` if the Error Module was initialized free of errors
`FALSE,` if an error occurred during initialization.

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

ICErrorExit()

The *ICErrorExit* function releases the resources reserved with *ICErrorInit* for error output. This function must be called once before the program is closed. There must be no further error outputs after this.

Visual C++ declaration (contained in the header file):

```
BOOL ICErrorExit()
```

Visual Basic declaration:

```
Declare Function ICErrorExit Lib “ICError.dll” () As Boolean
```

Call parameters:

None

Return value:

`TRUE,` if the Error Module was initialized free of errors
`FALSE,` if an error occurred during close down.

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

ICErrorMessage(...)

ICErrorMessage now signals an error. Whether this error appears in a window or is simply entered in the error archive depends on the error level set in the MOBIC Configurator and specified when the function was called.

Visual C++ declaration (contained in the header file):

```
BOOL ICErrorMessage(long lErrorLevel, long lMsgNum,
  LPCTSTR szParam1, LPCTSTR szParam2, LPCTSTR szParam3,
  LPCTSTR szParam4, LPCTSTR szParam5, LPCTSTR szParam6,
  LPCTSTR szParam7, LPCTSTR szParam8, LPCTSTR szParam9,
  LPCTSTR szParam10);
```

Visual Basic declaration:

```
Declare Function ICErrorMessage Lib "ICError.dll" _
  (ByVal lErrorLevel As Long, _
  ByVal lMsgNum As Long, ByVal szParam1 As String, _
  ByVal szParam2 As String, ByVal szParam3 As String, _
  ByVal szParam4 As String, ByVal szParam5 As String, _
  ByVal szParam6 As String, ByVal szParam7 As String, _
  ByVal szParam8 As String, ByVal szParam9 As String, _
  ByVal szParam10 As String ) As Boolean
```

Call parameters:

`lErrorLevel`
This parameter specifies the error level for error output. The following values must be used for the three possible levels:

Critical Error	(define: `IC_ERROR_CRITICAL`)	(=1)
Normal Error	(define: `IC_ERROR_NORMAL`)	(=2)
User Information	(define: `IC_ERROR_INFO`)	(=3)

In Visual C++, the levels defined in the header file should be used.

`lMsgNum`
This is the error ID that stands for the actual error. The error text along with the error ID is stored in an error text file (the error ID is a unique number identifying an error). The MOBIC is supplied with a standard error text file. You can also add your own error text files. The standard error text file contains standard error texts.
The standard error IDs and the corresponding error texts are listed at the end of this chapter.

`szParam1-szParam10`
The error texts from the error text file can include up to a maximum of 10 placeholders. `szParam1...10` are text strings that replace the relevant placeholders when the error is displayed. The placeholders in the error texts are “%1” to “%10”; `szParam1` replaces the placeholder “%, `szParam2` replaces the placeholder “%2” etc. If an error text does not use all 10 placeholders, 0 must be specified for the unused placeholders when the function is called (see Section 3.2.3).
The maximum length of an error output after the placeholders are replaced by text strings is 200 characters.

Return value:

`TRUE`, if error output was successful
`FALSE`, if an error occurred in this function.

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

ICErrorGetActiveLevel()

With ICErrorGetActiveLevel the error level currently set with the MOBIC Configurator can be obtained.

Visual C++ declaration (contained in the header file):
```
long ICErrorGetActiveLevel()
```

Visual Basic declaration:
```
Declare Function ICErrorGetActiveLevel Lib “ICError.dll” _
                                 () As Long
```

Call parameters:
None

Return value:
The error level set with the MOBIC Configurator

Critical Error	(define: `IC_ERROR_CRITICAL`)	(=1)
Normal Error	(define: `IC_ERROR_NORMAL`)	(=2)
User Information	(define: `IC_ERROR_INFO`)	(=3)

Note

If you change the error level, the relevant programs must be restarted.

3.2.2 Examples

Visual C++:

```
#include <windows.h>
#include “ICError.h”
#define APP_NAME TEXT(”MyApp”)

HANDLE OpenFileTest(LPCTSTR szFilename)
{
    HANDLE hFile;
    DWORD dwErrorNo;
    BOOL boErrMsgOK;
    LPTSTR lpMsgBuf;
    TCHAR  lpMsgNo[20];

    hFile = CreateFile(szFilename,GENERIC_READ, FILE_SHARE_READ, 0,
                       OPEN_EXISTING, FILE_ATTRIBUTE_READONLY, 0);

    if(hFile == INVALID_HANDLE_VALUE)
       {
       dwErrorNo = GetLastError();
       wsprintf(lpMsgNo, TEXT(”%d”), dwErrorNo);
       FormatMessage(  FORMAT_MESSAGE_ALLOCATE_BUFFER |
                       MAT_MESSAGE_FROM_SYSTEM |
                       FORMAT_MESSAGE_IGNORE_INSERTS,
                       NULL,
                       dwErrorNo,
                       0,
                       (LPTSTR) &lpMsgBuf,
                       0,
                       NULL);

    boErrMsgOK = ICErrorMessage(IC_ERROR_CRITICAL, 100000102,
                 szFilename, lpMsgNo, lpMsgBuf,0,0,0,0,0,0,0);
    LocalFree( lpMsgBuf );
    }
    else
    {
           CloseHandle(hFile);
    }
    return hFile;
}

int WINAPI WinMain ( HINSTANCE  hInstance,
                     HINSTANCE  hPrevInstance,
                     LPWSTR     lpCmdLine,
                     int        nCmdShow )
{
    if(ICErrorInit(APP_NAME) == FALSE)
    {
       MessageBox(0, TEXT(”ICErrorInit() reports error”),
       TEXT(”Error”), MB_OK);
       return 0;
    }

    OpenFileTest(TEXT(”test.txt”));

    if(ICErrorExit() == FALSE)
    {
       MessageBox(0, TEXT(”ICErrorExit() reports error”),
       TEXT(”Error”), MB_OK);
    }

    return 0;
}
```

Visual Basic:

```
Option Explicit

Declare Function ICErrorInit Lib “ICError.dll” () As Boolean
Declare Function ICErrorExit Lib “ICError.dll” () As Boolean
Declare Function ICErrorMessage Lib “ICError.dll” _
    (ByVal lErrorLevel As Long, ByVal lMsgNum As Long, _
     ByVal szParam1 As String, _
     ByVal szParam2 As String, _
     ByVal szParam3 As String, _
     ByVal szParam4 As String, _
     ByVal szParam5 As String, _
     ByVal szParam6 As String, _
     ByVal szParam7 As String, _
     ByVal szParam8 As String, _
     ByVal szParam9 As String, _
     ByVal szParam10 As String) _
   As Boolean

Const IC_ERROR_CRITICAL = 1
Const IC_ERROR_NORMAL = 2
Const IC_ERROR_INFO = 3


Private Sub Form_Load()
    ICErrorInit “MyApp”
End Sub


Private Sub ErrorButton_Click()
    ICErrorMessage IC_ERROR_CRITICAL, 100000102, _
    ”errormessage: parameter 1”, _
    ”errormessage: parameter 2”, _
    ”errormessage: parameter 3”, _
    0, 0, 0, 0, 0, 0, 0

End Sub
Private Sub Form_Unload(Cancel As Integer)
    ICTraceExit
    ICErrorExit
End Sub
```

3.2.3 Error IDs

The error IDs are 9-digit and structured as follows: The first four digits are the partner ID and the last five digits are the error number. This results in a unique error ID.

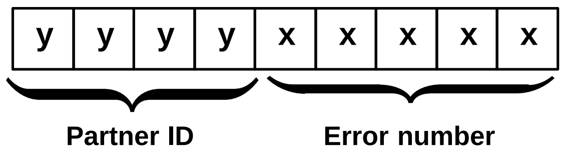


Figure 3-8 Structure of an Error ID

Standard error texts:

The MOBIC already has an error text file with the partner ID 1000 that contains standard errors only.

Whenever possible, you should use these standard error IDs.

Table 3-1 Error texts

Error ID	Text and Parameters
1000-00101	Error creating file ’%1’: Error number %2 - %3
1000-00102	Error opening file ’%1’: Error number %2 - %3
1000-00103	Error writing file ’%1’: Error number %2 - %3
1000-00104	Error deleting file ’%1’: Error number %2 - %3
1000-00201	File ’%1’ created
1000-00202	File ’%1’ opened
1000-00203	File ’%1’ closed
1000-00204	File ’%1’ deleted
1000-00301	Error creating registry entry ’%1’: Error number %2 - %3
1000-00302	Error reading registry entry ’%1’: Error number %2 - %3
1000-00303	Error writing registry entry ’%1’: Error number %2 - %3
1000-00304	Error deleting registry entry ’%1’: Error number %2 - %3
1000-00401	Program ’%1’ started
1000-00402	Program ’%1’ exited

Table 3-1 Error texts

Error ID	Text and Parameters
1000-00501	Error in %1 ADO record set: database: ’%2’ table: ’%3’ error number %4 - %5 *(As 1. parameters that pass the ADO function “Open”, “Up-date”, ... !)*
1000-01000	Error ’%1’ occurred, contact maintenance!

Creating your own error text file:

If you want to create your own error text file, select a partner ID in the range between 2000 and 2999.

Example: F2030_english.txt

In each line of the file, the text corresponding to an error number is specified.

“xxxxx” is the error number without the first four digits for the partner ID.

This is followed by the error text separated by a blank.

Use “\n” to specify a line break in the error text.

The parameters 1 to 10 are represented by the placeholders “%1” to “%10”.

The maximum length of an error text is 200 characters. Longer texts are truncated.

The error text file supplied with the MOBIC is stored in the OS flash memory. This ensures that the user cannot accidentally delete or modify this file.

Error text files added later are located in the battery-backed RAM area and are lost if both the main battery and backup battery are fully discharged.

When you install your application, all added error text files must be copied to the **`\Windows\LanguageFiles`** folder.

Example: **Error text file of partner ID 1000:**

```
Name:  F1000_english.txt
Contents:
00101 Error creating file ’%1’: Error number %2 - %3
00102 Error opening file ’%1’: Error number %2 - %3
00103 Error writing file ’%1’: Error number %2 - %3
00104 Error deleting file ’%1’: Error number %2 - %3
00201 File ’%1’ created
00202 File ’%1’ opened
.
.
.
```

3.3 Trace Module

The functionality of the Trace Module is in the DLL "**ICError.dll**". If you want to use these functions, just as with the Error Module, you must make the files ICError.lib and ICError.h known in the development environment (see Section 3.2).

3.3.1 Functions

ICTraceInit(...)

The *ICTraceInit* function initializes the Trace Module. It must be called once before the first trace output. We recommend that you call it right at the start of your program.

Visual C++ declaration (contained in the header file):

```
BOOL ICTraceInit(LPCTSTR szAppName, LPCTSTR szVersion)
```

Visual Basic declaration:

```
Declare Function ICTraceInit Lib "ICError.dll" _
              (ByVal szAppName As String, _
              ByVal szVersion As String) As Boolean
```

Call parameters:

`szAppName`
With this parameter, you can transfer the name of your program as a string. This name is displayed by the MOBIC Configurator. The application name must not exceed 32 characters in length.

`szVersion`
With this parameter, you can pass the version number of your program as a string. This version number appears in the first line of the trace file. The version must not exceed 32 characters.

Return value:
`TRUE,` if the Trace Module was initialized free of errors
`FALSE,` if an error occurred during initialization.

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

ICTraceExit()

The *ICTraceExit* function releases the resources reserved with *ICTraceInit* for trace output. This function must be called once before the program is closed. Following this, no further trace outputs are permitted.

Visual C++ declaration (contained in the header file):

```
BOOL ICTraceExit()
```

Visual Basic declaration:

```
Declare Function ICTraceExit Lib “ICError.dll” () As Boolean
```

Call parameters:
None

Return value:
`TRUE,` if the Trace Module could be uninitialized free of errors
`FALSE,` if an error occurred when uninitializing.

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

ICTraceGetActiveLevel()

With ICTraceGetActiveLevel, the trace level currently set with the MOBIC Configurator can be obtained.

Visual C++ declaration (contained in the header file):

```
long ICTraceGetActiveLevel()
```

Visual Basic declaration:

```
Declare Function ICTraceGetActiveLevel Lib “ICError.dll” _
        () As Long
```

Call parameters:
None

Return value:
0…3, the trace level set with the MOBIC Configurator

ICTraceMessage(...)

ICTraceMessage now creates a trace output. The output of the trace depends on the trace level set in the MOBIC Configurator and specified when the function is called.

The functions of the trace levels are described in Section 3.1.

Visual C++ declaration (contained in the header file):

```
BOOL ICTraceMessage (long lTraceLevel,
                     LPCTSTR szModulName,
                     LPCTSTR szFunctionName,
                     LPCTSTR szTraceText);
```

Visual Basic declaration:

```
Declare Function ICTraceMessage Lib “ICError.dll” _
                     (ByVal lTraceLevel As Long, _
                     ByVal szModulName As String, _
                     ByVal szFunctionName As String, _
                     ByVal szTraceText As String) As Boolean
```

Call parameters:

lTraceLevel
This parameter specifies the trace level for the trace output. The following values must be used for the three possible levels:

important trace	(define: IC_TRACE_IMPORTANT)	(=1)
extended trace	(define: IC_TRACE_EXTENDED)	(=2)
detailed trace	(define: IC_TRACE_DETAILED)	(=3)

In Visual C++, the levels defined in the header file should be used.

SzModulName
This string specifies the name of the module in which the trace output is made, for example, “myapp.cpp”. The module name must not exceed 32 characters in length.

szFunctionName
This string specifies the name of the function in which the trace output is made, the example, “WinMain.cpp”. The function name must not exceed 32 characters in length.

szTraceText
This string contains the text that will be output as the trace. The trace text can be up to a maximum of 200 characters long.

Return value:
`TRUE,` if trace output was successful
`FALSE,` if an error occurred in this function.

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

3.3.2 Examples

Visual C++:

```
#include <windows.h>
#include “ICError.h”

#define APP_NAME TEXT(”MyApp”)

#define APP_VERSION TEXT(”1.00”)

int WINAPI WinMain ( HINSTANCE  hInstance,
       HINSTANCE    hPrevInstance,
       LPWSTR       lpCmdLine,
       Int          nCmdShow )
{
    if(ICTraceInit(APP_NAME, APP_VERSION) == FALSE)
    {
       MessageBox(0, TEXT(”ICTraceInit() reports error”), TEXT(”Error”), MB_OK);
    return 0;
    }

    // allocate resources
    // ...
    // ...
    ICTraceMessage(IC_TRACE_DETAILED, TEXT(”myapp.cpp”), TEXT(”WinMain”),
                    TEXT(”resources are allocated”));

    // initialize variables
    // ...
    // ...
    ICTraceMessage(IC_TRACE_DETAILED, TEXT(”myapp.cpp”), TEXT(”WinMain”),
                    TEXT(”variables are initialized”));

    // ...
    // ...

    if(ICTraceExit() == FALSE)
    {
       MessageBox(0, TEXT(”ICTraceExit() reports error”), TEXT(”Error”), MB_OK);
    }

    return 0;
}
```

Visual Basic:

```
Option Explicit
Declare Function ICTraceInit Lib “ICError.dll” (ByVal szAppVersion, _
                ByVal version As String As Boolean
Declare Function ICTraceExit Lib “ICError.dll” () As Boolean
Declare Function ICTraceMessage Lib “ICError.dll” _
    (ByVal lTraceLevel As Long, _
     ByVal szModulName As String, _
     ByVal szFunctionName As String, _
     ByVal szTraceText As String) _
  As Boolean


Const IC_TRACE_IMPORTANT = 1
Const IC_TRACE_EXTENDED = 2
Const IC_TRACE_DETAILED = 3


Private Sub Form_Load()
    ICTraceInit “MyApp”, ”1.00”
End Sub


Private Sub TraceButton_Click()
    ICTraceMessage IC_TRACE_IMPORTANT, “myApp.cpp”, ”TraceButton_Click()”, _
                ”TraceButton clicked”
End Sub
Private Sub Form_Unload(Cancel As Integer)
    ICTraceExit
End Sub
```

3.4 Status Display

The MOBIC has four LEDs (read, green, yellow, yellow). Each LED can adopt one of the three states: on, off, or flashing. They are controlled by the functions of the MOBIC SDK listed below.

Table 3-2

	Red LED	**Green LED**	**Yellow LED**
lit	the device is supplied by an external power supply		user-defined
flashes	the main battery is almost empty	when a message is pending (for example, mail arrived)	user-defined
off	power supply from integrated battery	there is no message pending	user-defined

The functionality for the status display is in the DLL "**ICSDK.dll**". If you want to use this function, you must make the files ICSDK.lib and ICSDK.h known in the development environment. Follow the procedure as described in Section 3.2.

3.4.1 Functions

ICSetLED(...)

The *ICSetlED* function sets the state of the two yellow status LEDs.

Visual C++ declaration (contained in the header file):

```
BOOL ICSetLED(UINT Command, UINT Led, UINT OnPeriod,
                    UINT OffPeriod);
```

Visual Basic declaration:

```
Declare Function ICSetLED Lib "ICSDK.dll" _
                   (ByVal Command As Long, _
                   ByVal Led As Long, _
                   ByVal OnPeriod As Long, _
                   ByVal OffPeriod As Long) As Boolean
```

Call parameters:

`Command`
This parameter specifies the type of function for the required LED. The following values are possible:

LED permanently OFF	(define: ICSDK_LED_OFF)	(=0)
LED permanently ON	(define: ICSDK_LED_ON)	(=1)
LED flashes	(define: ICSDK_LED_FLASH)	(=2)
Status is inverted	(define: ICSDK_LED_INVERT)	(=3).

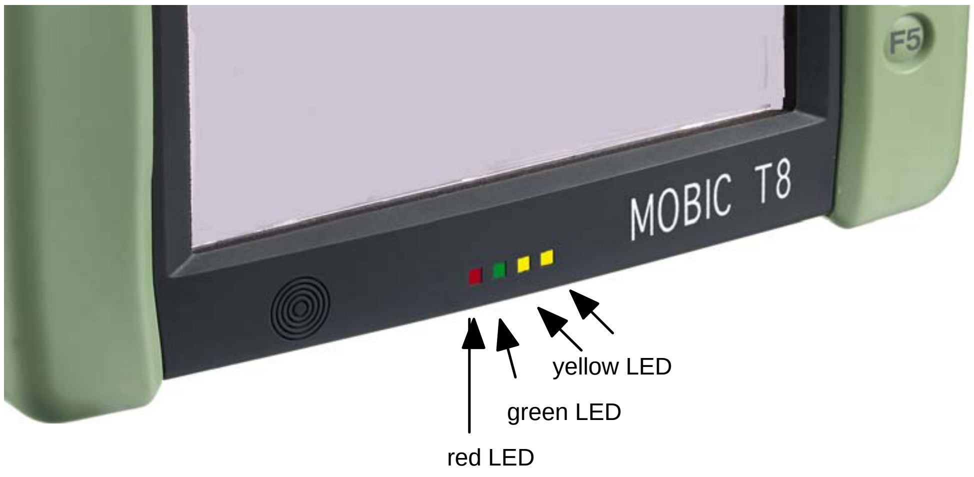


Figure 3-9 Status Displays

In Visual C++, you should use the constants defined for the function types.

`Led`
This parameter specifies which of the two LEDs will be activated. The following values are possible:

first yellow LED	(define: ICSDK_LED_YELLOW_1)	(=1)
second yellow LED	(define: ICSDK_LED_YELLOW_2)	(=2)

In Visual C++, you should use the function types defined in the header file.

`OnPeriod`
If you selected the function type “LED flashing” with the first parameter (`Command`), the OnPeriod parameter specifies the duration of the ON phase in steps of 1 millisecond. The system always rounds up to steps of 500 milliseconds.

Range of values: 1-65535

`OffPeriod`
If you selected the function type “LED flashing” with the first parameter (`Command`), the OffPeriod parameter specifies the duration of the OFF phase in steps of 1 millisecond. The system always rounds up to steps of 500 milliseconds.

Range of values: 1-65535

Return value:

`TRUE`, if the LEDs could be activated without errors
`FALSE`, if an error occurred activating the LEDs.

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

3.4.2 Examples

Visual C++:

```
#include <icsdk.h>
//...
if (ICSetLED(LED_FLASH, LED_YELLOW_1, 200, 1000) == FALSE)
{
    //...error
}
```

Visual Basic

```
Declare Function ICSetLED Lib “icsdk.dll” (ByVal Command As Long, _
                                   ByVal Led As Long, _
                                   ByVal OnPeriod As Long, _
                                   ByVal OffPeriod As Long) As Boolean

Const icLedYellow1 = 1
Const icLedYellow2 = 2
const icLedInvert = 3
Const icLedOff = 0
Const icLedOn = 1
Const icLedFlash = 2
Dim Result As Boolean

Result = ICSetLED (icLedFlash, icLedYellow1, 200, 1000)
```

3.5 Acoustic Signals

Over and above the standard Windows CE functions for outputting acoustic signals, the MOBIC SDK has an additional function for outputting acoustic signals. In contrast to the Windows CE functions, the MOBIC SDK function does not play an audio file but generates a tone with a selectable pitch and duration.

The functionality of the acoustic signals is in the DLL "**ICSDK.dll**". If you want to use this function, you must make the files ICSDK.lib and ICSDK.h known in the development environment. Follow the procedure as described in Section 3.2.

3.5.1 Functions

ICBeep(...)

The *ICBeep* function outputs a tone with a specific pitch and duration.

Visual C++ declaration (contained in the header file):

```
BOOL ICBeep (UINT Pitch, UINT Duration);
```

Visual Basic declaration:

```
Declare Function ICBeep Lib "icsdk.dll" _
                          (ByVal Pitch As Long, _
                          ByVal Duration As Long) As Boolean
```

Call parameters:

`Pitch`
This parameter specifies the pitch of the beep Hz.

Range of values: 100 - 5500

`Duration`
This parameter specifies the duration of the beep in milliseconds.

Range of values: 1 - 65535

Return value:
`TRUE,` if a tone could be activated without error
`FALSE,` if an error occurred.

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

3.5.2 Examples

Visual C++:

```
#include <icsdk.h>
//...
#define CONCERT_A 440
#define ONESECOND 1000
if (ICBeep(CONCERT_A, ONESECOND) == FALSE)
{
    //...error
}
```

Visual Basic

```
Declare Function ICBeep Lib “icsdk.dll” (ByVal Pitch As Long, _
                                         ByVal Duration As Long) As Boolean

Dim Result As Boolean
Result = ICBeep (440, 1000)
```

3.6 Setting the Brightness

The brightness of the display should normally be set using the MOBIC Configurator. If, on the other hand, you want to control of the brightness in your program, you can use this function.

The functionality for setting the brightness is in the DLL “**ICSDK.dll**”. If you want to use this function, you must make the files ICSDK.lib and ICSDK.h known in the development environment. Follow the procedure as described in Section 3.2.

3.6.1 Functions

ICAdjustLCDBrightness(...)

The *ICAdjustLCDBrightness* function can be used to query and set the brightness of the display. The brightness levels range from 1 to 31, level 1 being dark and 31 light.

Visual C++ declaration (contained in the header file):
```
BOOL ICAdjustLCDBrightness (UINT Command, UINT *Value);
```

Visual Basic declaration:
```
Declare Function ICAdjustLCDBrightness Lib “icsdk.dll” _
                   (ByVal Command As Long, _
                   ByRef Value As Long) As Boolean
```

Call parameters:

`Command`
This parameter specifies how the brightness will be set or queried:

Brightness one level lighter	(define: ICSDK_BRIGHTNESS_INCREASE)	(=0)
Brightness one level darker	(define: ICSDK_BRIGHTNESS_DECREASE)	(=1)
Reset brightness to level 16	(define: ICSDK_BRIGHTNESS_RESET)	(=2)
Set brightness to a defined level	(define: ICSDK_BRIGHTNESS_SET)	(=3)
Read out current brightness level	(define: ICSDK_BRIGHTNESS_GET)	(=4)

In Visual C++, the constants defined in the header file should be used.

`*Value`
Range of values: 1 - 31

Pointer to a variable of the type UINT. This parameter is only necessary for setting the brightness to a specific level (3) or for reading out the current level (4).
When you set the brightness to a specific level (3), the variable to which this pointer points must contain the value of the new brightness level.

When the current brightness level is read out, the value is written to this variable.

Return value:

`TRUE`, if no error occurred.
`FALSE`, if an error occurred.

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

3.6.2 Examples

Visual C++:

```
#include <icsdk.h>
//...
UINT Value;
if (ICAdjustLCDBrightness(ICSDK_BRIGHTNESS_GET, &Value) == FALSE)
{
    //...error
}
```

Visual Basic

```
Declare Function ICAdjustLCDBrightness Lib “icsdk.dll” (ByVal Command As Long, _
                                              ByRef Value As Long) As Boolean

Const icBrightnessIncrease = 0
Const icBrightnessDecrease = 1
Const icBrightnessReset = 2
Const icBrightnessSet = 3
Const icBrightnessGet = 4
Dim Result As Boolean
Dim Val As Integer
Result = ICAdjustLCDBrightness (icBrightnessGet, Val)
```

3.7 Turning off the LCD Backlighting

The backlighting of the display here is normally turned on and off by Windows CE. This setting can be made in the Windows CE Control Panel. The backlighting can be controlled with this function.

The functionality for turning off the LCD backlighting is in the DLL “**ICSDK.dll**”. If you want to use this function, you must make the files ICSDK.lib and ICSDK.h known in the development environment. Follow the procedure as described in Section 3.2.

3.7.1 Functions

ICLCDBacklightOff(...)

The *ICLCDBacklightOff* function can be used to turn the backlighting on and off.

Visual C++ declaration (contained in the header file):

```
BOOL ICLCDBacklightOff(BOOL bOff);
```

Visual Basic declaration:

```
Declare Function ICLCDBacklightOff Lib “icsdk.dll” _
                   (ByVal bOff As Long) As Boolean
```

Call parameters:

`bOff`
This parameter specifies whether the backlighting should be ON or OFF:

`TRUE,` - backlighting OFF
`FALSE,` - backlighting ON

Return value:
`TRUE,` no error
`FALSE,` error

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

3.7.2 Examples

Visual C++:

```
#include <icsdk.h>
//...
if (ICLCDBacklightOff(TRUE) == FLASE)
{
    //...error
}
```

Visual Basic

```
Declare Function ICLCDBacklightOff Lib “icsdk.dll” (ByVal bOff As Long) As Boolean

Dim Result As Boolean
Result = ICLCDBacklightOff (vbTrue)
```

3.8 Disabling the On/Off Button

Disabling and enabling the On/Off button should normally be done in the MOBIC Configurator. If, however, you require control of the On/Off button in your application, you can use this function.

The functionality for disabling the On/Off button is in the DLL "**ICSDK.dll**". If you want to use this function, you must make the files ICSDK.lib and ICSDK.h known in the development environment. Follow the procedure as described in Section 3.2.

3.8.1 Functions

ICProtectOffButton()

The I*CProtectOffButton* function can be used to disable or enable the On/Off button.

Visual C++ declaration (contained in the header file):

```
BOOL ICProtectOffButton (BOOL bLock);
```

Visual Basic declaration:

```
Declare Function ICProtectOffButton Lib "icsdk.dll" _
                   (ByVal bLock As Long) As Boolean
```

Call parameters:

`bLock`
This parameter specifies whether or not the On/Off button is disabled or enabled:

`TRUE,` On/Off button is disabled
`FALSE,`On/Off button is enabled

Return value:

`TRUE`, no error
`FALSE`, error

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

3.8.2 Examples

Visual C++:

```
#include <icsdk.h>
//...
if (ICProtectOffButton(TRUE) == 0)
{
    //...error
}
```

Visual Basic

```
Declare Function ICProtectOffButton Lib “icsdk.dll” (ByVal bLock As Long) _
                                                     As Boolean

Dim Result As Boolean
Result = ICProtectOffButton (vbTrue)
```

3.9 Reading the MAC Address of the MOBIC

Each MOBIC has a unique Ethernet MAC address. The MAC address can be read out with this function.

The functionality for reading out the MAC address is in the DLL “**ICSDK.dll**”. If you want to use this function, you must make the files ICSDK.lib and ICSDK.h known in the development environment. Follow the procedure as described in Section 3.2.

3.9.1 Functions

ICGetMACAddress (...)

The *ICGetMACAddress* function reads out the MAC address of the MOBIC. This MAC address is used by the OnBoard Ethernet.

Visual C++ declaration (contained in the header file):

```
BOOL ICGetMACAddress (BYTE MacAddress[6]);
```

Visual Basic declaration:

This function is not supported in Visual Basic.

Call parameters:

`MacAddress[6]`
The MAC address of the MOBIC here is entered here in this 6-byte array.

Return value:

`TRUE`, no error
`FALSE`, error

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

3.9.2 Examples

Visual C++

```
#include <icsdk.h>

unsigned char ucMacAddress[6];

if (ICGetMACAddress(ucMacAddress) == FALSE)
{
    //.... error
}
else
{
    TCHAR szMsg[255];
    wsprintf(szMsg, TEXT(”MAC Address: %02X:%02X:%02X:%02X:%02X:%02X\n”),
                 ucMacAddress [0], ucMacAddress [1], ucMacAddress [2],
                 ucMacAddress [3], ucMacAddress [4], ucMacAddress [5]);
    MessageBox(NULL, szMsg, TEXT(”MAC Address”), MB_OK);
}
```

3.10 Reading the Serial Number of the MOBIC

Each MOBIC has a unique serial number that cannot be modified. The serial number can be read out with this function.

The functionality for reading out the serial number is in the DLL “**ICSDK.dll**”. If you want to use this function, you must make the files ICSDK.lib and ICSDK.h known in the development environment. Follow the procedure as described in Section 3.2.

3.10.1 Functions

ICGetDeviceID(...)

The *ICGetDeviceID* function can be used to read out the unique serial number of the MOBIC.

<u>Visual C++ declaration (contained in the header file):</u>

```
BOOL ICGetDeviceID (BYTE SerNum[6]);
```

<u>Visual Basic declaration:</u>

This function is not supported in Visual Basic.

<u>Call parameters:</u>

`SerNum[6]`
The serial number of the MOBIC is entered here in this 6-byte array.

Return value:
`TRUE`, no error
`FALSE`, error

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

3.10.2 Examples

Visual C++:

```
#include <icsdk.h>

unsigned char ucSerNum[6];

if (ICGetDeviceID(ucSerNum) == FALSE)
{
    //.... error
}
else
{
    TCHAR szMsg[255];
    wsprintf(szMsg, TEXT(”Serial Number: %02X:%02X:%02X:%02X:%02X:%02X\n”),
                              ucSerNum[0], ucSerNum[1], ucSerNum[2],
                              ucSerNum[3], ucSerNum[4], ucSerNum[5]);
    MessageBox(NULL, szMsg, TEXT(”Unit DeviceID”), MB_OK);
}
```

3.11 Software Reset

With the Software Reset function, you can reset the MOBIC (warm reset).

The functionality for the Software Reset is in the DLL “**ICSDK.dll**”. If you want to use this function, you must make the files ICSDK.lib and ICSDK.h known in the development environment. Follow the procedure as described in Section 3.2.

3.11.1 Functions

ICResetDevice()

The *ICResetDevice* function executes any software reset on the MOBIC.

Visual C++ declaration (contained in the header file):

```
BOOL ICResetDevice (void);
```

Visual Basic declaration:

```
Declare Function ICResetDevice Lib “icsdk.dll” () As Boolean
```

Call parameters:

None

Return value:

`TRUE`, no error
`FALSE`, error

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

3.11.2 Examples

Visual C++:

```
#include <icsdk.h>

if (ICResetDevice() == FALSE)
{
    //.... error
}
```

Visual Basic

```
Declare Function ICResetDevice Lib “icsdk.dll” () As Boolean

Dim Result As Boolean
Result = ICResetDevice()
```

3.12 Function Keys

3.12.1 Setting the Reaction Time of the Function Keys (Debounce Time)

The reaction time of the function keys F1 to F5 (how long the key must be pressed) should normally be set in the MOBIC Configurator. If, however, you require control of the function keys in your application, you can use this function.

The functionality for setting the function keys is in the DLL “**ICSDK.dll**”. If you want to use this function, you must make the files ICSDK.lib and ICSDK.h known in the development environment. Follow the procedure as described in Section 3.2.

3.12.2 Functions

ICSetKeyDelay(...)

The *ICSetKeyDelay* function can be used to set the reaction time of the function keys.

Visual C++ declaration (contained in the header file):
```
BOOL ICSetKeyDelay (UINT Delay);
```

Visual Basic declaration:
```
Declare Function ICSetKeyDelay Lib “icsdk.dll” _
                   (ByVal Delay As Long) As Boolean
```

Call parameters:

`Delay`
Debounce time on the function keys in milliseconds.

Range of values: 0 - 500

Return value:

`TRUE`, no error
`FALSE`, error

In Visual C++, following an error, you can obtain an error code using the standard function *GetLastError()* that can be used for more detailed analysis of the error.

3.12.3 Examples

Visual C++:

```
#include <icsdk.h>

if (ICSetKeyDelay(50) == FALSE)
{
    //.... error
}
```

Visual Basic

```
Declare Function ICSetKeyDelay Lib “icsdk.dll” (ByVal Delay As Long) As Boolean

Dim Result As Boolean
Result = ICSetKeyDelay(50)
```

3.13 Function Key Assignment

If you want to assign different functions to the function keys for your application without changing the assignment of the function keys for other programs, you must make a manual modification in the registry.

This gives you the opportunity of assigning functions to the function keys for a specific program. This assignment is only active when the program is running in the foreground. The settings in the MOBIC Configurator are then ignored.

Registry Entry

[HKEY_LOCAL_MACHINE\SOFTWARE\Siemens\MOBIC\Config\
LaunchKeys\Apps\MyProgram]

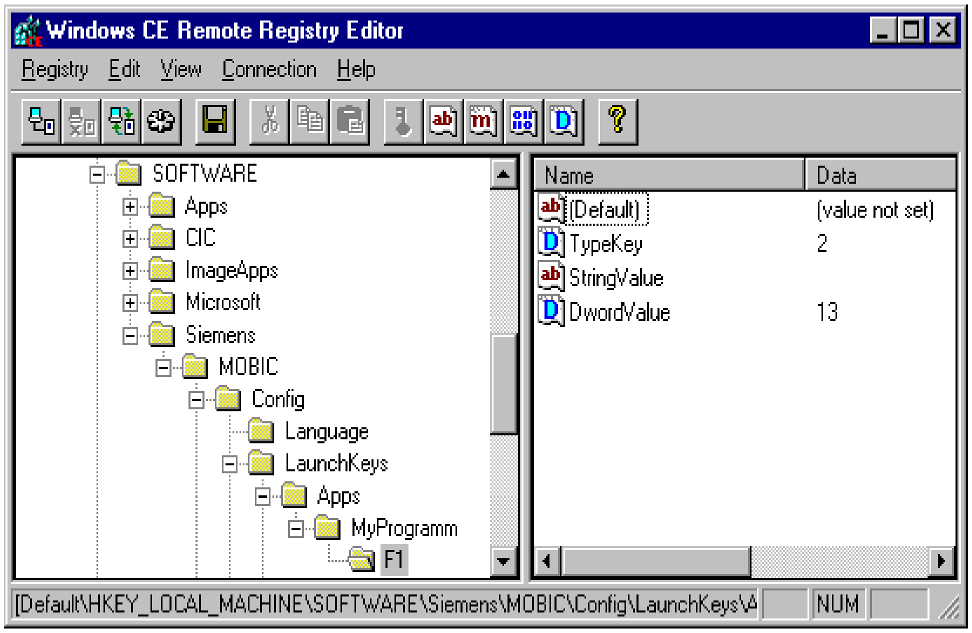
The name of the registry key (here "MyProgram") is the name of the application without the ".exe" extension. This mechanism is possible only with executable files (*.exe).

In this registry key, a further key with the name F1 to F5 can be created for each of the function keys.

If you do not require a different assignment for a function key, do not create a registry key for it.

Each of these keys must contain the following entries:

"TypeKey"	(type DWORD)
"StringValue"	(type STRING)
"DwordValue"	(type DWORD)



Name	Data
(Default)	(value not set)
TypeKey	2
StringValue	
DwordValue	13

[Default\HKEY_LOCAL_MACHINE\SOFTWARE\Siemens\MOBIC\Config\LaunchKeys\A

Figure 3-10

Permitted values:

”TypeKey”

0 = Start an application
1 = Display the software keyboard
2 = Trigger a keyboard event
3 = Display Windows help
4 = Prepare the right mouse click

”StringValue”

If “TypeKey”=0, the application name is entered here.

”DwordValue”

If “TypeKey”=2, the keycode is entered here.

If you want to use the function keys for your own application, you must make the setting for triggering a keyboard event for the required key (Typekey = 2)

When the function key is pressed, a corresponding keyboard event is reported to your application.

You must then simply program your application to react to the keyboard event.

Note

After modifying the registry, the device must be reset (warm reset) before the new function key assignment is adopted.

You can, for example, run a warm reset using the MOBIC Configurator (see also MOBIC manual).

Examples:

The **F1 key** is assigned the RETURN function (VK_RETURN=13) for the Pocket Word application:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Siemens\MOBIC\Config\
LaunchKeys\Apps\pword\F1]

”TypeKey”=dword:2

”StringValue”=””

”DwordValue”=dword:13 (0D hex)
```

The **F3 key** is assigned the function for displaying the software keyboard for the Calc application:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Siemens\MOBIC\Config\
LaunchKeys\Apps\Calc\F3]

”TypeKey”=dword:1

”StringValue”=””

”DwordValue”=dword:0
```

The **F5 key** is assigned the function of starting the calculator for the MyProgram application.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Siemens\MOBIC\Config\
LaunchKeys\Apps\MyProgram\F5]

”TypeKey”=dword:0

”StringValue”=”\\windows\\calc.exe”

”DwordValue”=dword:0
```

FAQs 4

How can I display traces via the serial interface?

To display traces via the COM port, you can, for example, use the Hyperterminal program of Windows NT.

1. Start the program.
2. Create a new connection and give it a suitable name.
3. Assign the required port to this connection, in
   Port Settings -> “Protocol” select “No protocol”
   and for -> “Bits per second” set the transmission rate.

Remember that the serial port you are using must not be used by any other program including Active Sync at the same time.

Can “Visual Studio” and “Windows CE Toolkit” be used ?

Before Microsoft brought out the “eMbedded Visual Tools 3.0”, the “Visual Studio” in conjunction with the “Windows CE Toolkit” had to be used. You should keep in mind the restrictions as described on the Internet:

http://support.microsoft.com/support/kb/articles/Q247/3/31.ASP

Recommendation:
We recommend that you use “eMbedded Visual Tools 3.0”.

How can I remove a program from the ”Application Trace Level” list of the MOBIC Configurator?

To remove a program from the ”Application Trace Level”, you must delete an entry (value) in the Registry using the Registry Editor.

The entry for the ”MyApp” application is as follows:
[HKEY_LOCAL_MACHINE\SOFTWARE\Siemens\MOBIC\TraceModule]
”MyApp_TraceLevel”=dword:0

To remove ”MyApp” from the ”Application Trace Level” list of the MOBIC Configurator, you must delete the ”MyApp_TraceLevel” entry.

How can I restore data automatically after a Cold Reset?

Backup and Restore
With the backup function, you can create a backup of your entire system. This contains all the programs and data you installed after receiving your device. A storage medium (PCMCIA card) is required for the backup. When you cold boot the MOBIC, you can, if you wish, restore a backup you made earlier.

The restore function allows you to restore a previously created backup manually. Restoring the backup on the MOBIC recreates the state as it was when you made the backup. In other words, programs and data installed after you made the backup are deleted!

Note

Backups created with a particular firmware version can only be restored to a MOBIC with the same firmware version.The detailed firmware version is entered in the 'Software' register of the MOBIC Configurator.

Backups also contain information on the touchscreen calibration. This means that if you restore a backup created on a different MOBIC, the touchscreen must be recalibrated. To calibrate the touchscreen, press F1 + F5 simultaneously.

Index