SIEMENS

SIMATIC

WinAC Slot T-Kit Version 3.3

Manual

Edition 01/2003

A5E00097452-02

Safety Guidelines

This manual contains notices intended to ensure personal safety, as well as to protect the products and connected equipment against damage. These notices are highlighted by the symbols shown below and graded according to severity by the following texts:

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 can result if proper precautions are not taken.

Caution

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

Notice

draws your attention to particularly important information on the product, handling the product, or to a particular part of the documentation.

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

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.

Trademarks

SIMATIC®, SIMATIC HMI® and SIMATIC NET® are registered trademarks of SIEMENS AG.

Third parties using for their own purposes any other names in this document which refer to trademarks might infringe upon the rights of the trademark owners.

Siemens AG
Bereich Automatisierungs- und Antriebstechnik
Geschaeftsgebiet Industrie-Automatisierungssysteme
Postfach 4848, D-90327 Nuernberg

Disclaim of Liability

We have checked the contents of this manual for agreement with the hardware and software described. Since deviations 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.

Preface

Purpose of this documentation

This documentation is aimed at original equipment manufacturers (OEMs) who wish to implement a high-speed data interface for a WinAC Slot 41x.

T-Kit is an application with which you, as an OEM, can implement a technological application to meet the requirements of your customers, which above all is designed for a simple and fast exchange of data between the CPU 41x-2 PCI and PC application.

Required basic skills

We assume the following basic skills for work to be possible with the documentation that follows:

- Extensive knowledge of WinAC Slot 41x
- Extensive knowledge of STEP 7
- Knowledge of C^{++}

Scope of the documentation

This documentation is applicable to T-Kit, version 3.3.

Changes compared to previous version

The following change has been made in comparison to the previous version of T-Kit, Version 3.2:

- T-Kit can also be operated under Windows XP Professional.

For your guidance

The documentation that follows is subdivided into the following subjects:

- Application of T-Kit
- System requirements and installation
- Getting started with an example program
- Data access classes for C^{++}
- Representation of data formats in SIMATIC S7

How the manual fits in

This manual describes how to use T-Kit. To work with T-Kit, you will require additional information on the following subjects:

Table 1-1 How the Manual Fits In

Manual/Manual Package	Contents
WinAC Slot 41x	Basic knowledge of CPUs 412-2 PCI and 416-2 PCI, and SIMATIC Computing.
Basic Grounding in STEP7 • *Getting Started and Exercises with STEP 7* • *Programming with STEP 7* • *Configuring Hardware and Connections with STEP 7* • *From S5 To S7, Conversion Guide*	The basic grounding for technical staff describing the procedures for implementing control tasks with STEP 7.
STEP7 reference • *LAD/FBD/STL Manual for S7-300/400* • *Standard and System Functions for S7-300/400*	Reference works describing the programming languages LAD, CSF and STL as well as standard and system functions additional to the *STEP 7* basic grounding.
S7-400 manuals • *S7-400, M7-400 Programmable Controllers Manual; Hardware and Installation* • *S7-400, M7-400 Programmable Controllers Reference Manual; Module Specifications*	The basic and reference knowledge referring to S7-400 CPUs, which is required for the CPU 412-2 PCI and CPU 416-2 PCI manuals.

Further Support

If you have any technical questions, please get in touch with your Siemens representative or agent responsible.

`http://www.siemens.com/automation/partner`

Training center

We offer a range of suitable courses to help get you started with the SIMATIC S7 automation system. Please contact your local training center or the central training center in Nuremberg, D-90327 Germany.

Phone: +49 (911) 895-3200.

http://www.sitrain.com

A&D Technical Support

Worldwide, available 24 hours a day:

Worldwide (Nuernberg) **Technical Support** 24 hours a day, 365 days a year Phone: +49 (0) 180 5050-222 Fax: +49 (0) 180 5050-223 E-Mail: adsupport@ siemens.com GMT: +1:00		
Europe / Africa (Nuernberg) **Authorization** Local time: Mon.-Fri. 7:00 to 17:00 Phone: +49 (0) 180 5050–222 Fax: +49 (0) 180 5050-223 E-Mail: adautorisierung@ siemens.com GMT: +1:00	**United States** (Johnson City) **Technical Support and Authorization** Local time: Mon.-Fri. 8:00 to 17:00 Phone: +1 (0) 423 262 2522 Fax: +1 (0) 423 262 2289 E-Mail: simatic.hotline@ sea.siemens.com GMT: –5:00	**Asia / Australia** (Beijing) **Technical Support and Authorization** Local time: Mon.-Fri. 8:30 to 17:30 Phone: +86 10 64 75 75 75 Fax: +86 10 64 74 74 74 E-Mail: adsupport.asia@ siemens.com GMT: +8:00
The languages of the SIMATIC Hotlines and the authorization hotline are generally German and English.		

Service & Support on the Internet

In addition to our documentation, we offer our Know-how online on the internet at:

`http://www.siemens.com/automation/service&support`

where you will find the following:

- The newsletter, which constantly provides you with up–to–date information on your products.
- The right documents via our Search function in Service & Support.
- A forum, where users and experts from all over the world exchange their experiences.
- Your local representative for Automation & Drives via our representatives database.
- Information on field service, repairs, spare parts and more under “Services”.

Contents

Figures

Tables

Product Overview

1

Introduction

The T-Kit is an interface (API) for technological applications that run with WinAC Slot 41x, **version 3.2 or higher**.

The T-Kit is intended for original equipment manufacturers (OEMs), who wish to implement a high-speed data interface between their application and CPU 41x-2 PCI.

The T-Kit consists of:

- T-Kit DLL (Dynamic Link Library)
- Header files
- Example code
- This documentation

Function

The T-Kit features the following functions for WinAC Slot 41x:

- Access from the STEP 7 user program to the dual-port RAM (input/output area) of the CPU 41x-2 PCI.
- Access from PC applications to the dual-port RAM of CPU 41x-2 PCI

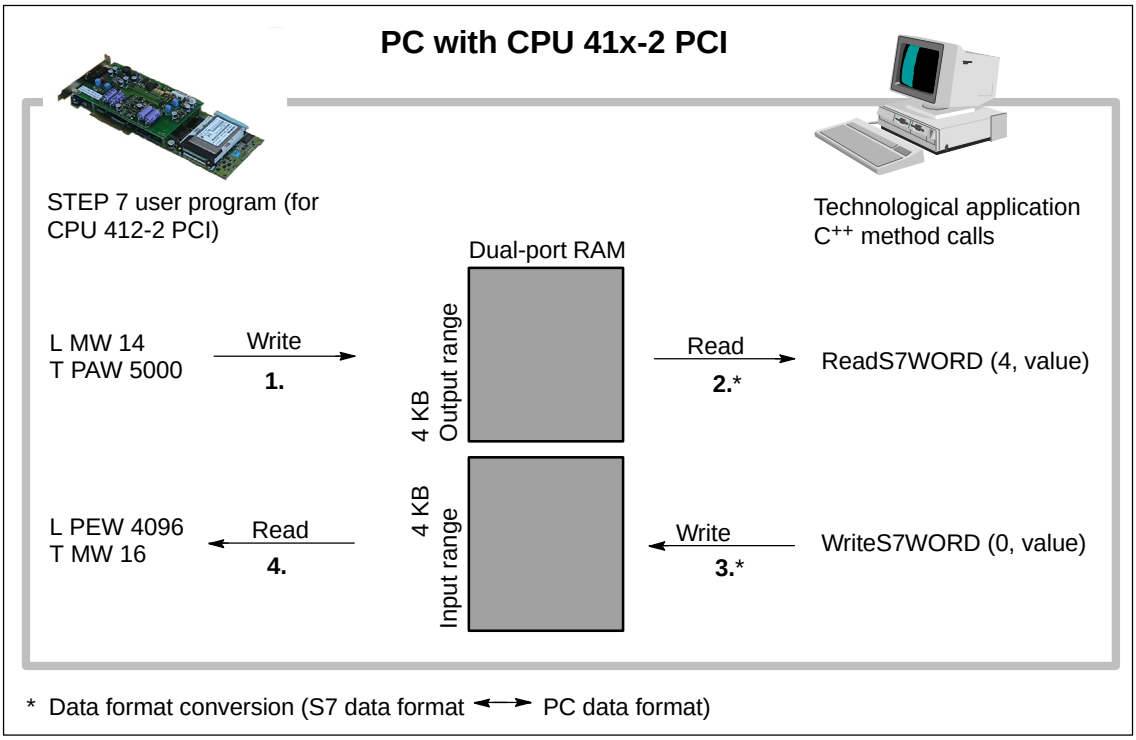

Figure 1-1 Theory of Operation of Data Exchange Between CPU 41x-2 PCI and Technological Application

Data exchange between CPU 41x-2 PCI and its technological application operates as follows:

1. You write the data required for the technological application in your STEP 7 user program by means of a transfer command to the dual-port RAM.
2. You read the data in your technological application by the polling method. To do this, you use the methods of the data access helper class (refer to Chapter 4).

   Result: The data are converted from the S7 data format to the PC data format that is used in Microsoft Windows.

Conversely, the data exchange from the technological application to CPU 41x-2 PCI is as follows:

3. You write the data of the technological application to the dual-port RAM using a method of the data access helper class (refer to Chapter 4).

   Result: The data are converted to the S7 data format and are available in the STEP 7 user program in S7 format.

4. Using the Load command, you read the data from the dual-port RAM within the STEP 7 user program.

Characteristics of T-Kit

The following initial conditions apply to the technological application, which you create with T-Kit:

PC side:

- Maximum data consistency is 4 byte. If you want to transmit large files consistently, you must ensure this by means of your technological application. For this you will find an example application on the T-Kit's CD in the Examples directory.
- The data access helper classes of the T-Kit support the following data types:
  - BOOL
  - BYTE, WORD, DWORD (double word)
  - INT (integer), DINT (integer, 32 Bit)
  - REAL (floating-point number)
  - S5TIME, TIME, DATE, TIME_OF_DAY
  - CHAR (character), STRING (of characters)

  You will find a detailed list of all data types supported by the T-Kit in Appendix A.
- Using the methods of the data access helper class, you automatically convert from the existing data type to the S7 data type, and the other way round.

CPU side:

- Only word and double-word accesses to **even** addresses are allowed.
- CPUs 41x-2 PCI each have 4 KB inputs and 4 KB outputs reserved in the dual-port RAM for the technological application:
  - CPU 412-2 PCI: PEW 4096 to 8190, PAW 4096 to 8190
  - CPU 416-2 PCI: PEW 16384 to 20478, PAW 16384 to 20478

Note

The above address areas are reserved in the CPU 41x-2 PCI for data exchange with a technological application – in other words, an I/O access error is not generated within this address area.

Exception: If the dual-port RAM is disabled by the PC when the PC runs up, for example, an I/O access error is generated.

Example applications

You will find example applications, which will assist you with the creation of your technological application, in the Examples directory on the “WinAC Slot T-Kit” CD-ROM.

System Requirements and Installation 2

Hardware requirements

For operating WinAC Slot 41x and T-Kit, we recommend the following hardware:

- WinAC Slot 41x Version 3.2 or higher
- PC with
  - Pentium processor clocking at least 300 MHz
  - Not less than 128 MB RAM
  - Windows 2000 Professional with SP3 or higher or Windows XP Professional with SP1 or higher or Windows NT Version 4 with SP6 or higher.
- A color monitor, keyboard, and mouse (or other pointing device) that are supported by Microsoft Windows NT
- A hard disk with at least 40 MB of spare storage space

Software requirements

To develop a technological application with T-Kit, you will require the following software packages, which must be installed on your PC:

- WinAC Slot 41x, version 3.2 or higher
- WinAC Slot T-Kit, version 3.3
- STEP 7, version 5.2
- Microsoft Visual Developers Studio (Visual C^{++}), Version 6, Service Pack 3 or lhigher

Lincense conditions

A single WinAC Slot T-Kit license is required for each **development workstation**.

Distribution of T-Kit applications

The WinAC Slot T-Kit includes a license for the installation on a development workplace.

Applications created with the WinAC Slot T-Kit require no additional licensing from Siemens. In other words, you may freely reproduce and use such applications. Additional licenses for the WinAC Slot T-Kit are therefore unnecessary.

To operate your T-Kit applications on a destination PC with WinAC Slot without having to install the WinAC Slot T-Kit itself, you may copy the following DLL together with the T–Kit application:

\WINNT\System32\S7TKIT_DLL.DLL

To activate the S7TKIT_DLL.DLL on the destination PC, simply place the DLL in the \WINNT\System32 directory of the destination PC. It is not necessary to enter the DLL in the registry.

Ideally, this task should be part of a setup program created for your T-Kit application.

Installation

The WinAC Slot T-Kit software includes a setup feature for each CPU type that performs automatic installation.

The Setup program guides you step by step through the installation process. You can switch to the next step or to the previous step from any position. To start the installation program, proceed as follows:

1. Insert the CD in your CD-ROM drive.
2. Double-click the “setup.exe” file to select it.

Once the installation has been completed successfully, a message to that effect is displayed on the screen.

Getting Started with an Example Program 3

Overview

With T-Kit you received examples, located in the Examples directory, of how to use T-Kit for creating a technological application in.

The following example is intended to help you understand the theory of how T-Kit functions.

Contents of the example

The following example demonstrates how you exchange a word between CPU 41x-2 PCI and the technological application. You have the following dialog box for this on your PC:

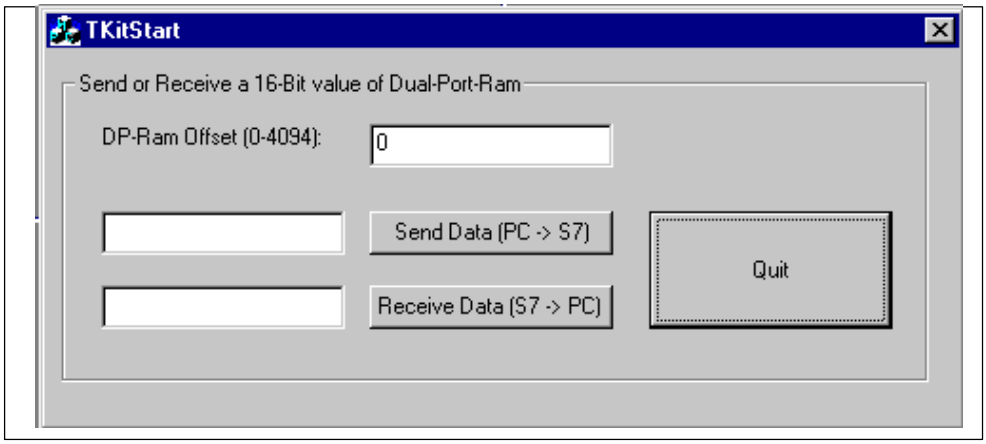

Figure 3-1 TKitStart Dialog Box

DP-Ram Offset (0 – 4094): Here you enter the offset address of the dual-port RAM from and to which the “Word” type value has to be read and written.

Send Data (PC –> S7): Here you enter the value that you want to have sent from your PC to CPU 41x-2 PCI and click on “Send Data (PC –> S7)” to confirm.

Receive Data (PC –> S7): Click on “Receive Data (PC –> S7) to receive the value that has been transferred from CPU 41x-2 PCI to your PC.

Quit: “Quit” closes the TKitStart dialog box.

Example program (starting)

The example program consists of the following files:

- STEP 7 user program consisting of OB 1.

  Depending on the type of CPU, a ”Word” type value is read from CPU 41x-2 PCI or written to CPU 41x-2 PCI in OB 1.

- C++ program consisting of header, source code and resources files (refer to Figure 3-3).

  The files show you how to load a ”Word” type value from CPU 41x-2 PCI or transfer it to CPU 41x-2 PCI from your PC. The majority of the files are created automatically by Visual C++ Studio; only those files are shown below which have been expanded for the example.

STEP 7 user program

Below, you will find the structure of OB 1 as a function of the type of CPU.

Table 3-1 STEP 7 User Program (Example of CPU 412-2 PCI)

```
STL                          Explanation
OB1                                  Example of CPU 412-2 PCI
       L      PIW    4096
       T      PQW    4096
```

Table 3-2 STEP 7 User Program (Example of CPU 416-2 PCI)

```
STL                          Explanation
OB1                                  Example of CPU 416-2 PCI
       L      PIW    16384
       T      PQW    16384
```

Procedure for using the example program

To work with the example program, you must proceed as follows:

1. Start Visual C^{++}.
2. Open the workspace by choosing **File > Open Workspace**.

   Result: The following window opens:

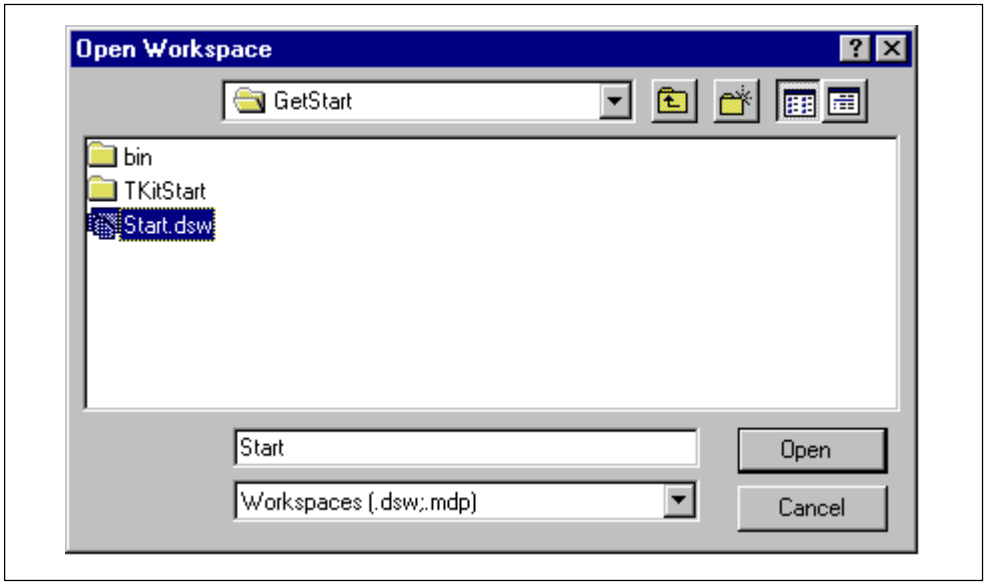

Figure 3-2 Open Workspace

3. Select the Start.dsw file and then click **Open**.

4. Create the TKitStart.exe file by choosing the command **Create > Create TKitStart.exe**.

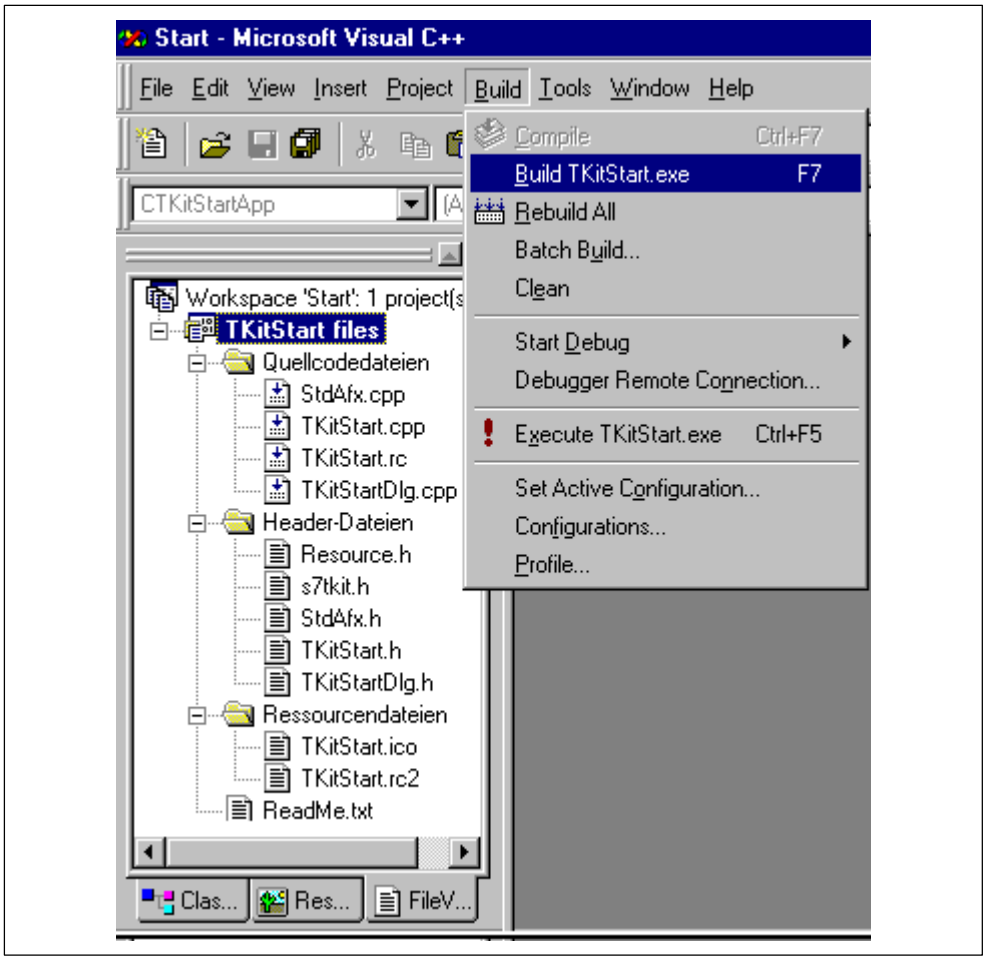

Figure 3-3 Create the TKitStart.exe File

5. In SIMATIC Manager, set the access point of the application to “PC internal” by choosing **Options > Set PG/PC Interface**.
6. Load the highlighted OB1 in SIMATIC Manager (in the Blocks folder) into CPU 41x-2 PCI.
7. Using Explorer, start the “TKitStart.exe” file.
8. Change CPU 41x-2 PCI to “RUN”.

   Result: You can now send data to CPU 41x-2 PCI or receive data from CPU 41x-2 PCI in the “TKitStart” window.

C++ files

You will find a simple C++ listing below. The passages relevant to T-Kit have a gray background.

Note

At the end of the C++ listing you must always delete the pointer (pointer release) so that subsequent PC applications can work properly.

Header file s7tkit.h

The s7tkit.h header file is a requirement for being able to use the data access helper class:

Table 3-3 Header File s7tkit.h

C-Listing

```
// s7tkit.h :
//
// The following ifdef block is the standard way of creating macros which make
// exporting from a DLL simpler. All files within this DLL are compiled with the
// S7TKIT_EXPORTS symbol defined on the command line. this symbol should not be
// defined on any project that uses this DLL. This way any other project whose source
// files include this file see S7TKIT_API functions as being imported from a DLL,
// wheras this DLL sees symbols defined with this macro as being exported.


#ifdef S7TKIT_EXPORTS
#define S7TKIT_API __declspec(dllexport)
#else
#define S7TKIT_API __declspec(dllimport)
#endif

////////////////////////////////////////////////////////////////////////////
// Type definitions
////////////////////////////////////////////////////////////////////////////

typedef short int          SINT16;
typedef DWORD              BIT32;
typedef WORD               BIT16;
typedef int                SINT32;
typedef BYTE               BIT8;
typedef unsigned short int UINT16;
typedef BYTE               UINT8;
```

Table 3-3 Header File s7tkit.h (continued)

C-Listing

```
////////////////////////////////////////////////////////////////////////////
// Data Access Helper Class
////////////////////////////////////////////////////////////////////////////

class S7TKIT_API CWinAcReadData // Read-Only
{
  public:
    // Customer methods
    CWinAcReadData();
    ~CWinAcReadData();   bool ReadS7BOOL(long byteOffset, int bitNo, bool &value);
    bool ReadS7BYTE(long byteOffset, BIT8 &value);
    bool ReadS7CHAR(long byteOffset, char &value);
    bool ReadS7DATE(long byteOffset, UINT16 &value);
    bool ReadS7DINT(long byteOffset, SINT32 &value);
    bool ReadS7DWORD(long byteOffset, BIT32 &value);
    bool ReadS7INT(long byteOffset, SINT16 &value);
    bool ReadS7REAL(long byteOffset, float &value);
    bool ReadS7S5TIME(long byteOffset, BIT16 &value);
    bool ReadS7STRING(long byteOffset, UINT8 ReadMax, char* string);
    bool ReadS7STRING_LEN(long byteOffset, UINT8 &maxLen, UINT8 &curLen);
    bool ReadS7TIME(long byteOffset, SINT32 &value);
    bool ReadS7TIME_OF_DAY(long byteOffset, UINT32 &value);
    bool ReadS7WORD(long byteOffset, BIT16 &value);

   // Internal
   protected:
    HANDLE hDriver;
    BYTE*  pDPR;
    int    Status;
    bool   OffsetCheck(long byteOffset, int size);
};


class S7TKIT_API CWinAcReadWriteData:public CWinAcReadData // Read/Write
{
  public:

    // Customer methods
    CWinAcReadWriteData();

    bool WriteS7BOOL(long byteOffset, int bitNo, bool &value);
    bool WriteS7BYTE(long byteOffset, BIT8 &value);
    bool WriteS7CHAR(long byteOffset, char &value);
    bool WriteS7DATE(long byteOffset, UINT16 &value);
    bool WriteS7DINT(long byteOffset, SINT32 &value);
    bool WriteS7DWORD(long byteOffset, BIT32 &value);
    bool WriteS7INT(long byteOffset, SINT16 &value);
    bool WriteS7REAL(long byteOffset, float &value);
    bool WriteS7S5TIME(long byteOffset, BIT16 &value);
    bool WriteS7STRING(long byteOffset, char* string);
    bool WriteS7TIME(long byteOffset, SINT32 &value);
    bool WriteS7TIME_OF_DAY(long byteOffset, UINT32 &value);
    bool WriteS7WORD(long byteOffset, BIT16 &value);

  };
```

Header file TKitStartDlg.h

In der Header-Datei TKitStartDlg.h rufen Sie die Header-Datei s7tkit.h auf und greifen auf den Pointer pInterface zu:

Table 3-4 Header File TKitStartDlg.h

C-Listing

```
// TKitStartDlg.h :
//

#if
!defined(AFX_TKITSTARTDLG_H__3FF60425_332C_11D5_BB11_08000624AC1F__INCLUDED_)
#define AFX_TKITSTARTDLG_H__3FF60425_332C_11D5_BB11_08000624AC1F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

        ////////////////////////////////
        // START, own code Getting Started //
        ////////////////////////////////

        #include "s7tkit.h"

        ////////////////////////////////
        // END , own code Getting Started  //
        ////////////////////////////////


/////////////////////////////////////////////////////////////////////////////
// CTKitStartDlg

class CTKitStartDlg : public CDialog
{
public:
        CTKitStartDlg(CWnd* pParent = NULL);

        //{{AFX_DATA(CTKitStartDlg)
        enum { IDD = IDD_TKITSTART_DIALOG };
        CString m_Str_ValueS7PC;
        CString m_Str_ValuePCS7;
        long    m_long_Offset;
        //}}AFX_DATA

        //{{AFX_VIRTUAL(CTKitStartDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);

        //}}AFX_VIRTUAL
```

Table 3-4 Header File TKitStartDlg.h (continued)

C-Listing

```
protected:
        HICON m_hIcon;

                ////////////////////////////////
                // START, own code Getting Started //
                ////////////////////////////////

                CWinAcReadWriteData *pInterface;

                ////////////////////////////////
                // END , own code Getting Started  //
                ////////////////////////////////



        //{{AFX_MSG(CTKitStartDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        afx_msg void OnButtonReceiveS7Pc();
        afx_msg void OnButtonSendPcS7();
        afx_msg void OnQuit();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}


#endif //
!defined(AFX_TKITSTARTDLG_H__3FF60425_332C_11D5_BB11_08000624AC1F__INCLUDED_)
```

Implementation file TKitStartDlg.cpp

In the TKitStartDlg.cpp implementation file you instance the data access helper class and read or write a ”Word” type value.

Table 3-5 Implementation File TKitStartDlg.cpp

C-Listing

```
// TKitStartDlg.cpp :
//

#include ”stdafx.h”
#include ”TKitStart.h”
#include ”TKitStartDlg.h”

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);
        //}}AFX_VIRTUAL

protected:
        //{{AFX_MSG(CAboutDlg)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}
```

Table 3-5 Implementation File TKitStartDlg.cpp (continued)

C-Listing

```
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTKitStartDlg

CTKitStartDlg::CTKitStartDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CTKitStartDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CTKitStartDlg)
        m_Str_ValueS7PC = _T("");
        m_Str_ValuePCS7 = _T("");
        m_long_Offset = 0;
        //}}AFX_DATA_INIT
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CTKitStartDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CTKitStartDlg)
        DDX_Text(pDX, IDC_EDIT_VALUE_RECEIVE_S7_PC, m_Str_ValueS7PC);
        DDX_Text(pDX, IDC_EDIT_VALUE_SEND_PC_S7, m_Str_ValuePCS7);
        DDX_Text(pDX, IDC_EDIT_OFFSET_HEX, m_long_Offset);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CTKitStartDlg, CDialog)
        //{{AFX_MSG_MAP(CTKitStartDlg)
        ON_WM_SYSCOMMAND()
        ON_WM_PAINT()
        ON_WM_QUERYDRAGICON()
        ON_BN_CLICKED(IDC_BUTTON_RECEIVE_S7_PC, OnButtonReceiveS7Pc)
        ON_BN_CLICKED(IDC_BUTTON_SEND_PC_S7, OnButtonSendPcS7)
        ON_BN_CLICKED(ID_QUIT, OnQuit)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTKitStartDlg

BOOL CTKitStartDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);
```

Table 3-5 Implementation File TKitStartDlg.cpp (continued)

C-Listing

```
        CMenu* pSysMenu = GetSystemMenu(FALSE);
        if (pSysMenu != NULL)
        {
                CString strAboutMenu;
                strAboutMenu.LoadString(IDS_ABOUTBOX);
                if (!strAboutMenu.IsEmpty())
                {
                        pSysMenu->AppendMenu(MF_SEPARATOR);
                        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
                }
        }

        SetIcon(m_hIcon, TRUE);                         //
        SetIcon(m_hIcon, FALSE);                        //

                ////////////////////////////////
                // START, own code Getting Started //
                ////////////////////////////////

                 pInterface = new CWinAcReadWriteData();
                // while instancing you build a link to the DPRAM

                ////////////////////////////////
                // END , own code Getting Started  //
                ////////////////////////////////


        return TRUE;
}

void CTKitStartDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
                CAboutDlg dlgAbout;
                dlgAbout.DoModal();
        }
        else
        {
                CDialog::OnSysCommand(nID, lParam);
        }
}
```

Table 3-5 Implementation File TKitStartDlg.cpp (continued)

C-Listing

```
void CTKitStartDlg::OnPaint()
{
        if (IsIconic())
        {
                CPaintDC dc(this);

                SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;

                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
        }
}

HCURSOR CTKitStartDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}



void CTKitStartDlg::OnButtonReceiveS7Pc()
{

        long  offset;
        bool success;

        BIT16 bit16 = 0x00;

        UpdateData(TRUE);

        offset = m_long_Offset;

        success = pInterface->ReadS7WORD(offset, bit16);

        if (success)
        {
                m_Str_ValueS7PC.Format("%04X", bit16);
        }
        else
        {
                m_Str_ValueS7PC.Format("Nothing");

        }
```

Table 3-5 Implementation File TKitStartDlg.cpp (continued)

C-Listing

```
        UpdateData(FALSE);

}



void CTKitStartDlg::OnButtonSendPcS7()
{

        long offset;
        bool success;

        BIT16 bit16 = 0x00;

        UpdateData(TRUE);

        offset = m_long_Offset;

    LPSTR str = m_Str_ValuePCS7.GetBuffer(10);

        sscanf( str, "%x", &bit16 );

        success = pInterface->WriteS7WORD(offset, bit16);

}

void CTKitStartDlg::OnQuit()
{

        delete pInterface;

    // Very Important. When the pointer is not deleted ,you are not calling the
    // destructor of the data helper class and you crash the operating system

        CDialog::OnCancel();

}
```

Data Access Classes

4

In this chapter

Introduction

The data access helper classes include the CWinACReadData and CWinACReadWriteData helper classes.

Note

The read and write methods were split in order to ensure basic security. This makes sure that you cannot inadvertently overwrite data when you call the read method.

Use the data access class methods to exchange data between the STEP 7 user program and your technological application:

- These methods help you to avoid programming errors such as access to offsets outside the valid area or writing of invalid pointers.
- In addition, you perform the necessary byte exchange by converting the data from “big endian” format (used in SIMATIC S7) into “little endian” format (used in Windows).

Assignment of the addresses between the CPU 41x-2 PCI and dual-port RAM

The following tables show you the relationship between the addresses in the STEP 7 user program and the addresses in the dual-port RAM.

Table 4-1 Address Assignment between CPU 412-2 PCI and Dual-Port RAM

Address in the CPU 412-2 PCI	Offset (Parameters for Methods of Data Access Class)	Address in Dual-Port RAM
I/O output word 4096 to 8190	0000 to 0FFF	C000 to CFFF (read area)
I/O output word 4096 to 8190	0000 to 0FFF	E000 to EFFF (write area)

Table 4-2 Address Assignment bewteen CPU 416-2 PCI and Dual-Port RAM

Address in the CPU 416-2 PCI	Offset (Parameters for Methods of Data Access Class)	Address in Dual-Port RAM
I/O input word 16384 to 20478	0000 to 0FFF	3C000 to 3CFFF (read area)
I/O input word 16384 to 20478	0000 to 0FFF	3E000 to 3EFFF (write area)

Example: You wish to send a word from the CPU 416-2 PCI to the technological application. If you write the word by means of the load command “T PW 16384” to the dual-port RAM, you can read the word to offset 0000 from the dual-port RAM with the help of one of the following methods (read method).

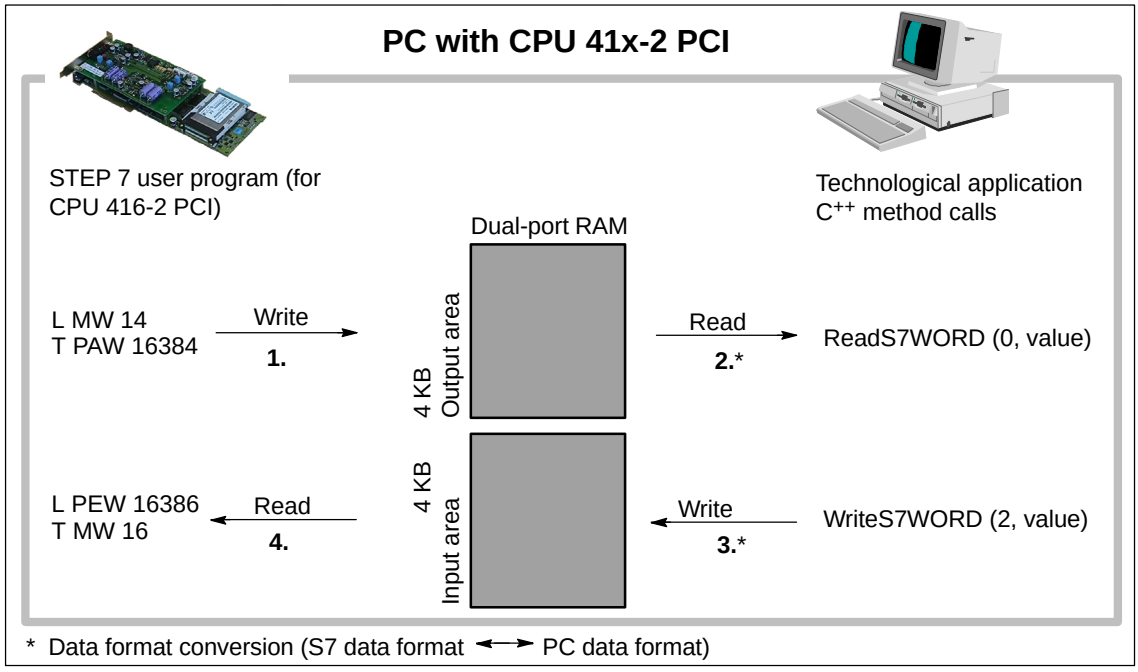

Figure 4-1 Theory of Operation of Data Exchange Between CPU 41x-2 PCI and Technological Application

Cycle overflow from continous load

If you create a continuous load by using the data access classes of the T-Kit (e.g. invoking loops with 10000 ”WriteS7Word”), WinAC Slot may go into the ”Stop” operating mode. The reason for this is that the PC processor generates a continuous data stream through the PCI bus. This, in turn, means that the dual-port RAM between the PC and WinAC–Slot is continuously resevered for the PC. The end result is that WinAC Slot changes to the “Stop” operating mode or invokes OB 122 because of the cycle time overflow. You should therefore avoid continuous loads, perhaps by inserting “Sleeps” or using a lower repetition rate for you loops.

Problem

```
for (i=0;i<10000,i++)
{
 bResult=InOut.WriteS7Word(0, Bit16Val);
}
```

Solution

```
for (k=1;0<10;k++)
{
  for (i=0;i<1000,i++)
  {
    bResult=InOut.WriteS7Word(0, Bit16Val);
  }
  sleep(100);
}
```

4.1 CWinACReadData Helper Class

In this chapter

Introduction

The CWinACReadData helper class fetches input data from the read area of the dual-port RAM, which were stored there in the STEP 7 user program using Transfer commands.

Every method follows the following format:

```
ReadS7<datatype>(long byteOffset, <datatype>& value)
```

All the following methods have the return value True or False, depending on whether the read method was successful or not. For example, the read method can prove unsuccessful if the offset is outside the valid area on the dual-port RAM.

The methods automatically perform all byte format conversions that are necessary between the original byte format and the internal WinAC byte format.

<datatype>: Name of S7 data type. You will find more information about S7 data types in Appendix A.

byte offset: Start address (in bytes) of the value in the dual-port RAM in the read area. (For example, the 4th double word begins at byte offset 12.)

value: Destination address of the data needing to be stored.

4.1.1 bool ReadS7Bool(long byteOffset, int bitNo, bool &value) method

This method retrieves the requested bit value from the dual-port RAM (read area) *byteOffset*at *byteOffset* and stores the value in the *value* parameter.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (read area)
- *int bitNo*: bit number to be retrieved (starting from right to left)
- *bool &value*: value of the bit to be retrieved

4.1.2 bool ReadS7BYTE(long byteOffset, BIT8 &value) method

This method retrieves a byte (8 bits) from the dual-port RAM (read area) at *byteOffset* and stores the value in the *value* parameter.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (read area)
- *BIT8 &value*: value of the retrieved byte

4.1.3 bool ReadS7WORD(long byteOffset, BIT16 &value) method

This method retrieves a 16-bit value from the dual-port RAM (read area) and stores it as an unsigned 16-bit integer in the *value* parameter.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (read area)
- *BIT16 &value*: value of the retrieved data

4.1.4 bool ReadS7DWORD(long byteOffset, BIT32 &value) method

This method retrieves a 32-bit double word from the dual-port RAM (read area) at *byteOffset* and stores the value as an unsigned 32-bit integer in the *value* parameter.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (read area)
- *BIT32 &value*: value of the retrieved data

4.1.5 bool ReadS7INT(long byteOffset, SINT16 &value) method

This method retrieves 16 bits from the dual-port RAM (read area) at *byteOffset* and stores the value as a signed 16-bit integer in the *value* parameter.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (read area)
- *SINT16 &value*: value of the retrieved data

4.1.6 bool ReadS7DINT(long byteOffset, SINT32 &value) method

This method retrieves 32 bits from the dual-port RAM (read area) at *byteOffset* and stores the value as a signed integer in the *value* parameter.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (read area)
- *SINT32 &value*: value of the retrieved data

4.1.7 bool ReadS7REAL(long byteOffset, float &value) method

This method retrieves 32 bits from the dual-port RAM (read area) at *byteOffset* and stores the value as a floating-point number in the *value* parameter.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (read area)
- *float &value*: value of the retrieved data

4.1.8 bool ReadS7S5TIME(long byteOffset, Bit16 &value) method

This method retrieves the a 16-bit value from the dual-port RAM (read area) at *byteOffset* and stores the data in the *value* parameter.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (read area)
- *BIT16 &value*: value of the retrieved data

4.1.9 bool ReadS7TIME(long byteOffset, SINT32 &value)

This method retrieves a 32-bit value from the dual-port RAM (read area) at *byteOffset* and stores it as a signed 32-bit integer in the *value* parameter.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (read area)
- *SINT32 &value*: value of the retrieved data

4.1.10 bool ReadS7DATE(long byteOffset, UINT16 &value) method

This method retrieves a 16-bit value from the dual-port RAM (read area) at *byteOffset* and stores the value as an unsigned 16-bit integer (S7 data type: Date) in the *value* parameter.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (read area)
- *UINT16 &value*: value of the retrieved data

4.1.11 bool ReadS7TIME_OF_DAY(long byteOffset, UINT32 &value) method

This method retrieves a 32-bit value from the dual-port RAM (read area) at *byteOffset* and stores it as an unsigned 32-bit integer in the *value*) parameter.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (read area)
- *UINT32 &value*: value of the retrieved data

4.1.12 bool ReadS7CHAR(long byteOffset, char &value) method

This method retrieves an 8-bit character from the dual-port RAM (read area) at *byteOffset* and stores the value in the *value* parameter.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (read area)
- *char &value*: value of the retrieved character

4.1.13 bool ReadS7STRING(long byteOffset, UINT8 readMax, char* string) method

This method retrieves a string from the dual-port RAM (read area), starting with *byteOffset*. This methods continues retrieval until all characters in the string have been retrieved or retrieved up to the *readMax* character. readMax cannot be longer than the maximum string length. This can be determined by ReadS7STRING_LEN. The string is stored in the memory, to which *string* is pointing.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (read area)
- *UINT8 readMax*: maximum number of characters to be retrieved
- *char* string*: memory in which the string is stored

4.1.14 bool ReadS7STRING_LEN(long byteOffset, UINT8 &maxLen, UINT8 &curLen) method

The method reads the information about the length of a string. The maximum length of the string is stored in the *maxLen* parameter. The current length of the string is stored in the *curLen* parameter.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (read area)
- *UINT8 maxLen*: maximum length of the string
- *UINT8 &curLen*: current length of the string

4.2 CWinACReadWriteData Helper Class

In this chapter

Introduction

The CWinACReadWriteData helper class expands the CWinACReadData class by additional methods for writing data as S7 data types to the write area of the dual-port RAM. You then access the data in the write area of the dual-port RAM in the STEP 7 user program with load commands.

Every method follows the following format:

```
WriteS7<datatype>(long byteOffset, <datatype>& value)
```

All the following methods have the return value True or False, depending on whether the write method was successful or not. For example, the write method can prove unsuccessful if the offset is outside the valid area on the dual-port RAM.

The methods automatically perform all byte format conversions that are necessary between the original byte format and the internal WinAC byte format.

<datatype>: Name of S7 data type. You will find more information about S7 data types in Appendix A.

byte offset: Start address (in bytes) of the value in the dual-port RAM in the write area.

value: data that are required to be stored in the memory

Note

The CWinACReadData helper class bequeaths all methods to the CWinACReadWriteData helper class, so that all read methods that appear in the CWinACReadData helper class are also available in the CWinACReadWriteData helper class.

4.2.1 bool WriteS7BOOL(long byteOffset, int bitNo, bool &value) method

This method sets a bit on the value stored in the *value* parameter. The position of the bit is determined by *bitNo*, the address of the byte in the *long byteOffset* parameter

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (write area)
- *int bitNo*: bit number to be written (starting from right to left)
- *bool &value*: value of the bit to be written

4.2.2 bool WriteS7BYTE(long byteOffset, BIT8 &value) method

This method writes the data stored in the *value* parameter to a byte (8 bits) in the dual-port RAM (write area) at the *byteOffset* position.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (write area)
- *BIT8 &value*: value of the byte to be written

4.2.3 bool WriteS7WORD(long byteOffset, BIT16 &value) method

This method writes the 16-bit value stored in the *value* parameter to the dual-port RAM (write area) at the *byteOffset* position.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (write area)
- *BIT16 &value*: value of the data to be written

4.2.4 bool WriteS7DWORD(long byteOffset, BIT32 &value) method

This method writes the 32-bit double word stored in the *value* parameter to the dual-port RAM (write area) at the *byteOffset* position.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (write area)
- *BIT32 &value*: value of the data to be written

4.2.5 bool WriteS7INT(long byteOffset, SINT16 &value) method

This method writes the signed 16-bit integer value stored in the *value* parameter to the dual-port RAM (write area) at the *byteOffset* position.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (write area)
- *SINT16 &value*: value of the data to be written

4.2.6 bool WriteS7DINT(long byteOffset, SINT32 &value) method

This method writes the 32-bit value stored in the *value* parameter to the dual-port RAM (write area) at the *byteOffset* position.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (write area)
- *SINT32 &value*: value of the data to be written

4.2.7 bool WriteS7REAL(long byteOffset, float &value) method

This method writes a 32-bit floating-point number stored in the *value* parameter to the dual-port RAM (write area) at the *byteOffset* position.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (write area)
- *float &value*: value of the data to be written

4.2.8 bool WriteS7S5TIME(long byteOffset, BIT16 &value) method

This method writes the 16-bit value stored in the *value* parameter to the dual-port RAM (write area) at the *byteOffset* position.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (write area)
- *BIT16 &value*: value of the data to be written

4.2.9 bool WriteS7TIME(long byteOffset, SINT32 &value) method

This method writes the 32-bit value stored in the *value* parameter to the dual-port RAM (write area) at the *byteOffset* position.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (write area)
- *SINT32 &value*: value of the data to be written

4.2.10 bool WriteS7DATE(long byteOffset, UINT16 &value) method

This method writes the 16-bit value stored in the *value* parameter to the dual-port RAM (write area) at the *byteOffset* position.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (write area)
- *UINT16 &value*: value of the character to be written

4.2.11 bool WriteS7TIME_OF_DAY(long byteOffset, UINT32 &value) method

This method writes the 32-bit value stored in the *value* parameter to the dual-port RAM (write area) at the *byteOffset* position.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (write area)
- *UINT32 &value*: value of the data to be written

4.2.12 bool WriteS7CHAR(long byteOffset, char &value) method

This method writes the 8-bit character stored in the *value* parameter to the dual-port RAM (write area) at the *byteOffset* position.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (write area)
- *char &value*: value of the character to be written

4.2.13 bool WriteS7STRING(long byteOffset, char* string) method

This method writes a string to the dual-port RAM (write area) beginning at *byteOffset*. The method expects the string to end at ZERO ('\0'). If the whole string does not fit into the space intended for it (for example, if the current length is longer than the maximum length or there is not sufficient space in the dual-port RAM (write area), the write method fails. For a string with an odd length (in bytes), a byte is automatically added when this method is used.

Return value:

- *bool*: success or fail (true = success)

Parameters:

- *long byteOffset*: offset in byte in the dual-port RAM (write area)
- *char* string*: pointer to a string that ends on ZERO.

SIMATIC S7 Data Types

A

In this chapter

A.1 Data Types in SIMATIC S7

Elementary data types

The following table shows you the elementary data types in SIMATIC S7, which you can access with the help of T-Kit:

Table A-1 Elementary Data Types in SIMATIC S7

Type and Description	Size in Bits	Format Options	Area and Number Representation (Lowest to Highest Value)	Example:
BOOL (Bit)	1	Bool text	TRUE/FALSE	True
BYTE (Byte)	8	Hexadecimal number	B16#0 to B16#FF	L B#16#10 L byte#16#10
WORD (Word)	16	Pure binary number	2#0 to 2#1111_1111_1111_1111	L 2#0001_0000_0000_0000
		Hexadecimal number	W#16#0 to W#16#FFFF	L W#16#1000 L word16#1000
		BCD	C#0 to C#999	L C#998
		Unsigned decimal number	B#(0,0) to B#(255,255)	L B#(10,20) L byte#(10,20)
DWORD (Double Word)	32	Pure binary number	2#0 to 2#1111_1111_1111_1111_ 1111_1111_1111_1111	2#1000_0001_0001_1000_ 1011_1011_0111_1111
		Hexadecimal number	DW#16#0000_0000 to DW#16#FFFF_FFFF	L DW#16#00A2_1234 L dword#16#00A2_1234
		Unsigned decimal number	B#(0,0,0,0) to B#(255,255,255,255)	L B#(1, 14, 100, 120) L byte#(1,14,100,120)
INT (Integer)	16	Signed decimal number	-32768 to 32767	L 1
DINT (Integer, 32 Bits)	32	Signed decimal number	L#-2147483648 to L#2147483647	L L#1
REAL (Floating-Point Number)	32	IEEE floating-point number	Upper limit: ± 3.402823e+38 Lower limit: ±1.175 495e-38	L 1.234567e+13
S5TIME (SIMATIC Time)	16	S7 time in steps of 10 ms (default value)	S5T#0H_0M_0S_10MS to S5T#2H_46M_30S_0MS und S5T#0H_0M_0S_0MS	L S5T#0H_1M_0S_0MS L S5TIME#0H_1H_1M_0S_0MS
TIME (IEC Time)	32	IEC time in steps of 1 ms, signed integer	T#24D_20H_31M_23S_648MS to T#24D_20H_31M_23S_647MS	L T#0D_1H_1M_0S_0MS L TIME#0D_1H_1M_0S_0MS

Table A-1 Elementary Data Types in SIMATIC S7 (continued)

Type and Description	Size in Bits	Format Options	Area and Number Representation (Lowest to Highest Value)	Example:
DATE (IEC Date)	16	IEC date in steps of one day	D#1990-1-1 to D#2168-12-31	L D#1994-3-15 L DATE#1994-3-15
TIME_OF_DAY (Time)	32	Time of day in steps of 1 ms	TOD#0:0:0.0 to TOD #23:59:59.999	L TOD#1:10:3.3 L TIME_OF_DAY#1:10:3.3
CHAR (Character)	8	ASCII character	'A', 'B', etc.	L 'E'

Complex data types

STRING: Defines a group of not more than 254 characters (CHAR data type). The standard area reserved for a string of characters consists of 256 bytes. This is the space required to store 254 characters and a header of 2 bytes. You can reduce the storage space for a string of characters by defining the number of characters which you want to have saved in the string for example: string[7] 'Siemens').

A.2 Format of the WORD and DWORD Data Types for Binary-Coded Decimal Numbers

The binary-coded decimal (BCD) represents a decimal number by means of groups of binary digits (bits). A group of 4 bits represent a digit of a decimal number of the sign of the decimal number. The groups of 4 bits form one word (16 bits) or double word (32 bits).

The four highest-order bits specify the sign of the number (“1111“ means minus and “0000“ means plus). Commands with BCD-coded operands evaluate only the highest-order bit (15 for word format and 31 for format).

The following table shows the format and area for both types of BCD numbers.

Table A-2 Format and Area of the WORD and DWORD Data Types

Format	Area
Word (16 bits, 3-digit BCD number, signed)	-999 to +999
Double Word (32 bits, 7-digit BCD number, signed)	-9 999 999 to +9 999 999

Word format

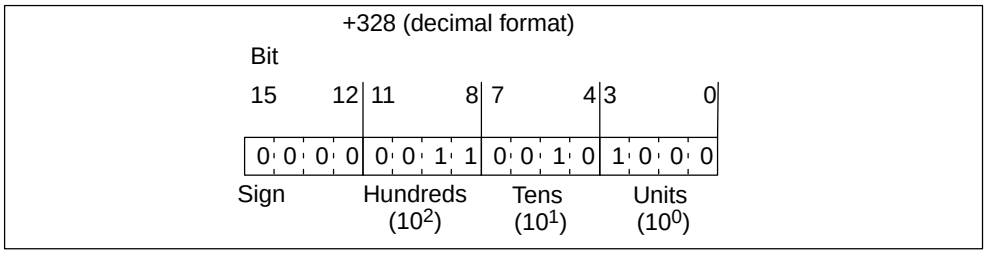

Figure A-1 Word Format

Double word format

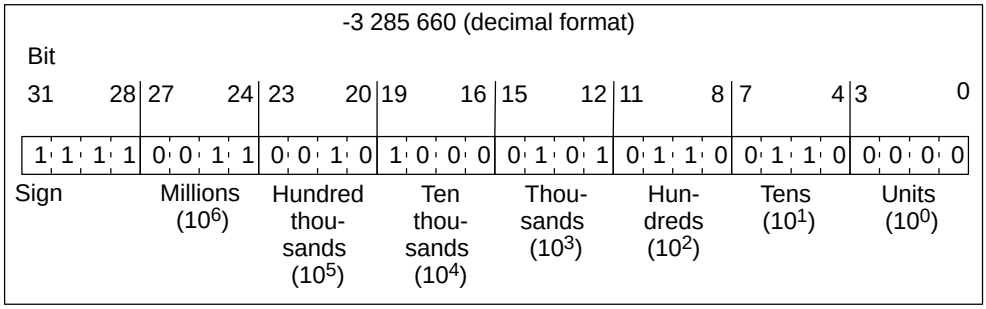

Figure A-2 Double Word Format

A.3 Format of the INT Data Type (16-bit integer)

An integer has a sign that indicates whether the integer is a positive or negative number. The space that an integer (16 bits) takes up in the memory is one word. The following table shows the area of an integer (16 bits).

Table A-3 Format and Area of the INT Data Type

Format	Area
Integer	-32 768 to +32 767

INT

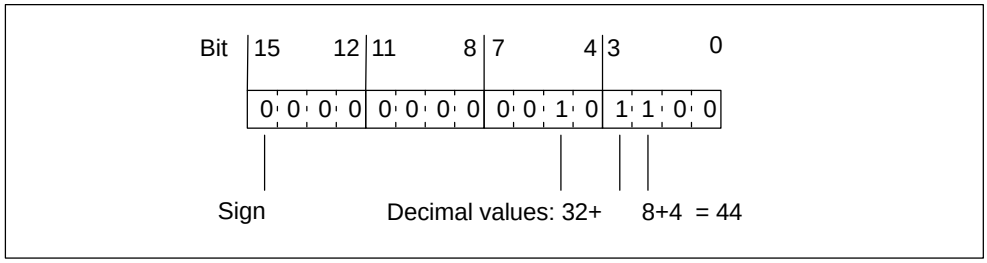

Figure A-3 DINT Format Represented as Pure Binary Number

A.4 Format of the DINT Data Type (32-bit integer)

An integer has a sign that indicates whether the integer is a positive or negative number. The space that an integer (32 bits) takes up in the memory is two words. The following table shows the area of an integer (32 bits).

Table A-4 Format and Area of the DINT Data Type

Format	Area
Integer (32 bits)	-2 147 483 648 to +2 147 483 647

DINT

The following figure shows the integer –500 000 as a pure binary number. In the binary number system, the negative form of an integer is shown as the two's complement of the positive integer. You obtain the two's complement of an integer by reversing the signal states of all bits and adding +1 to the result.

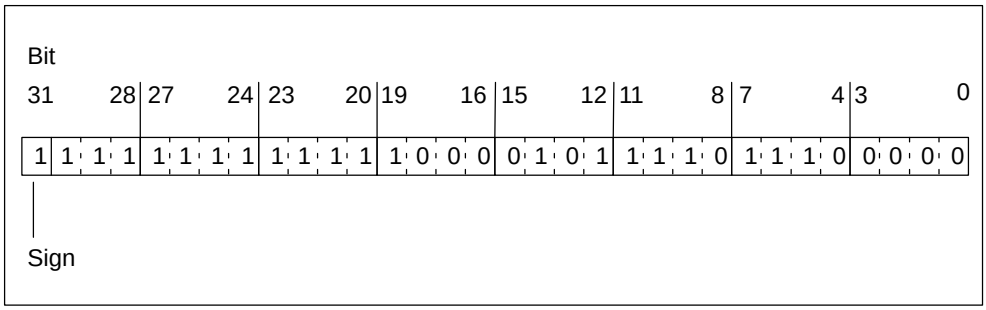

Figure A-4 DINT Format Represented as Pure Binary Number

A.5 Format of the REAL Data Format (Floating-Point Number)

Numbers in floating-point number format are shown in the general form of "number = m * b to the power of E". The base "b" and exponent "E" are integers, the mantissa "m" is a rational number.

This number representation has the advantage of it being possible to display very large and very small values in a limited space. A further range of numbers can be covered with the limited number of bits for the mantissa and exponent.

The disadvantage is the limited accuracy of calculation: for example, when forming the sum of two numbers the exponents have to be matched by moving the mantissa – hence floating decimal point (addition of the mantissae of two numbers having the same exponent).

Floating-point number format in STEP 7

Floating-point numbers in STEP 7 conform to the basic format, single width, described in the ANSI/IEEE Standard 754-1985, IEEE Standard for Binary Floating-Point Arithmetic. They consist of the following components:

- The sign V
- The exponent e = E + bias, increased by a constant (bias = +127)
- The fractional part of the mantissa m.

  The whole number part of the mantissa is not stored with the rest, because it is always equal to 1 within the valid number range.

The three components together occupy one double word (32 bits):

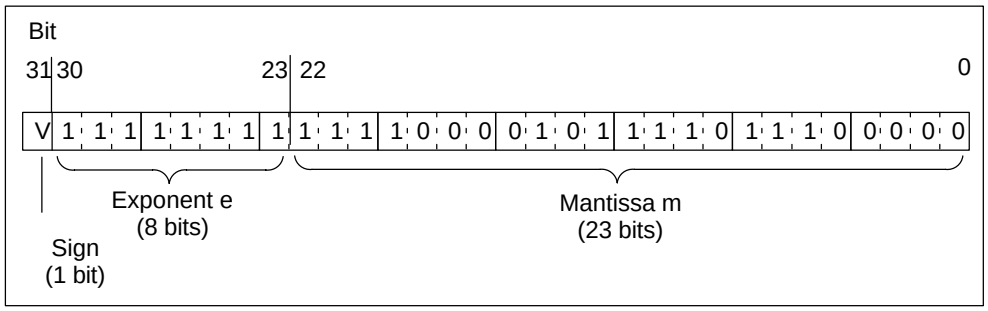

Figure A-5 Representation of the Data Format

The following table shows the values of the individual bits in floating-point format.

Table A-5 REAL Data Type: Values of the Individual Bits

Component of the Floating-Point Number	Bit Number	Value
Sign V	31	
Exponent e	30	2^7
...	...	...
Exponent e	24	2^1
Exponent e	23	2^0
Mantissa m	22	2^{-1}
...	...	...
Mantissa m	1	2^{-22}
Mantissa m	0	2^{-23}

Using the three components **V**, **e** and **m**, the value of a number represented in this form is defined by the formula:

Number = 1.**m** * 2 $^{(e- \text{ bias})}$

Where:

- e: 1 ⩽ e 254
- Bias: bias = 127. This means that an additional sign is not required for the exponent.
- V: for a positive number, V = 0 and for a negative number, V = 1.

Value range of floating-point numbers

Using the floating-point format shown above, the following results:

- The smallest floating-point number = 1.0 * 2 to the power of (1-127) = 1.0 * 2 to the power of (-126)
  = 1.175 495E-38 and
- The largest floating-point number = 2-2 to the power of (-23) * 2 to the power of (254-127) = 2-2 to the power of (-23) * 2 to the power of (+127) = 3.402 823E+38

The number zero is represented with e = m = 0; e = 255 and m = 0 stands for ”infinite.”

Table A-6 Format and Area of the REAL Data Type

Format	Area
Floating-point numbers conforming to ANSI/IEEE Standard	-3.402 823E+38 to -1.175 495E-38 and 0 and +1.175 495E-38 to +3.402 823E+38

The next table shows the signal state of the bits in the status word for the results of operations with floating-point numbers that do not lie within the valid range.

Table A-7 Signal State of Bits in the Status Word When Results with Floating-Point Numbers Do Not Lie within the Valid Range

Invalid Range for a Result	1 A	A0	OV	OS
-1.175494E-38 < result < -1.401298E-45 (negative number) underflow	0	0	1	1
+1.401298E-45 < result < +1.175494E-38 (positive number) underflow	0	0	1	1
Result < -3.402823E+38 (negative number) overflow	0	1	1	1
Result < -3.402823E+38 (positive number) overflow	1	0	1	1
Not a valid floating-point number or invalid instruction (input value outside the valid value range)	1	1	1	1

Note when using mathematical operations:

The result ”Not a valid floating-point number” is obtained, for example, when you attempt to extract the square root from -2. You should therefore always evaluate the status bits first in math operations before continuing calculations based on the result.

Note in the case of “Force Variables”:

If the values for floating-point operations are stored in memory double words, for example, you can modify these values with any bit patterns. However, not every bit pattern is a valid number.

Accuracy when calculating floating-point numbers

Caution

Extensive calculations with numbers exhibiting very large differences (several orders of magnitude) can produce inaccurate results.

The floating-point numbers in STEP 7 are accurate to 6 decimal places. You can therefore only specify a maximum of 6 decimal places when entering floating-point constants.

Note

The calculation accuracy of 6 decimal places means, for example, that the addition of number1 + number2 = number1 if number1 is greater than number 2 * 10 to the power of ^{y}, where y > 6:

100 000 000 + 1 = 100 000 000.

Examples of numbers in floating-point format

The following figure shows the floating-point format for the following decimal values:

- 10.0
- p (3.141593)
- Square root of 2 (p2 = 1.414214)

The number **10.0** in the first example results from its floating-point format (hexadecimal representation: 4120 0000) as follows:

e = $2^1 + 2^7 = 2 + 128 = 130$

m = $2^{-2} = 0.25$

This results in: $1.m * 2^{(e - Bias)} = 1.25 * 2^{(130 - 127)} = 1.25 * 2^3 = 10.0$.

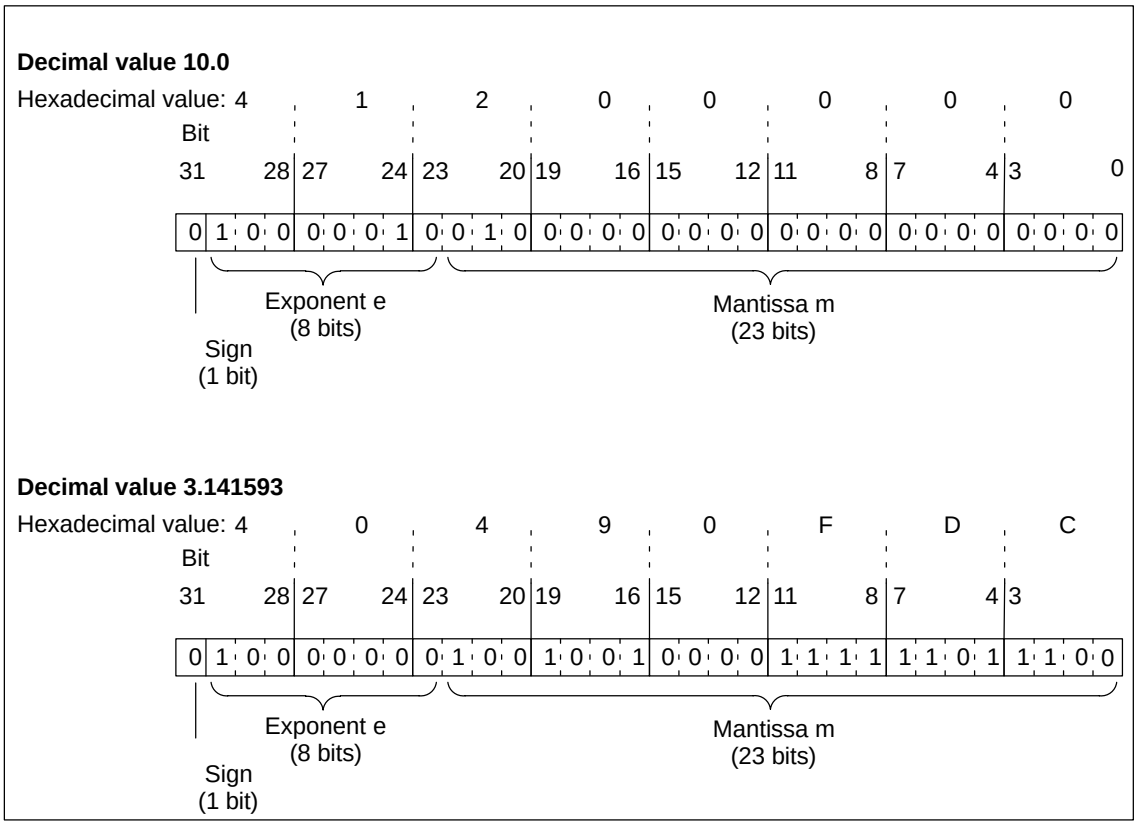

Figure A-6 Examples of the REAL Data Type

A.6 Format of the S5TIME Data Type (time duration)

When you enter time duration using the S5TIME data type, your entries are stored in binary coded decimal format. The following figure shows the content of the time address with a time value of 127 and a time base of 1 s.

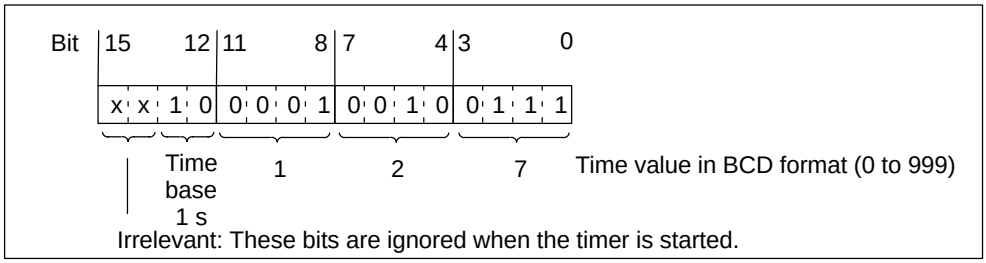

Figure A-7 Examples of the S5TIME Data Type

When working with S5TIME, you enter a time value in the range of 0 to 999 and you indicate a time base (see the following table). The time base indicates the interval at which a timer decrements the time value by one unit until it reaches 0.

Table A-8 Time Base for S5TIME

Time Base	Binary Code for Time Base
10 ms	00
100 ms	01
1 s	10
10 s	11

You can pre-load a time value using either of the following syntax formats:

- L W#16#wxyz

  Where: w = the time base (in other words, time interval or resolution)

  xyz = the time value in binary coded decimal format

- L S5T#aH_bbM_ccS_dddMS

  Where a = hours, bb = minutes, cc = seconds and ddd = milliseconds.

  The time base is selected automatically and the value is rounded to the next lower number with that time base.

The maximum time value that you can enter is 9,990 seconds, or 2H_46M_30S.

A.7 Format of the TIME Data Type

A variable with TIME (time duration) data type takes up a double word. The representation contains the details for days (d), hours (h), minutes (m), seconds (s) and milliseconds (ms), it being possible to omit specific details. The contents of the variable will be interpreted as milliseconds (ms) and stored as a signed 32-bit fixed-point number.

You do not have to specify all units of time (for example, T#5h10s is valid).

If only one unit is specified, the absolute value for days, hours and minutes must not exceed the upper or lower limit values.

T#–65535 and T#+65535 are the upper and lower limit values for seconds and milliseconds.

If more than one unit of time is specified, the unit

- hours must not exceed a value of 23,
- minutes must not exceed a value of 59,
- seconds must not exceed a value of 59,
- milliseconds must not exceed a value of 999.

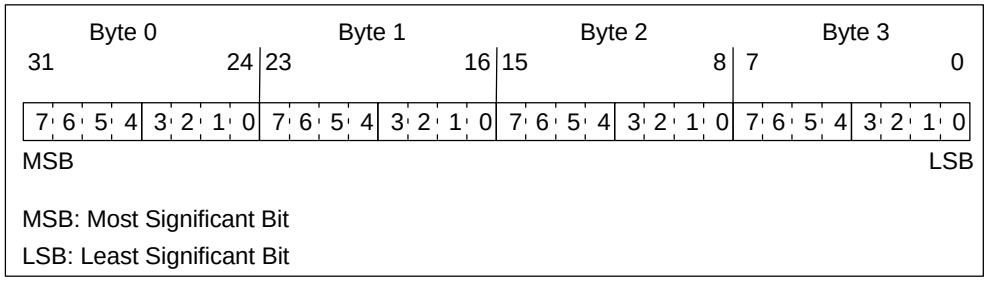

Figure A-8 Examples of the TIME Data Type

A.8 Format of the DATE Data Type

A variable with DATE (date) data type is stored in a word as an unsigned fixed-point number. The contents of the variable corresponds to the number of days since January 1, 1990. The representation contains the year, the month and the day.

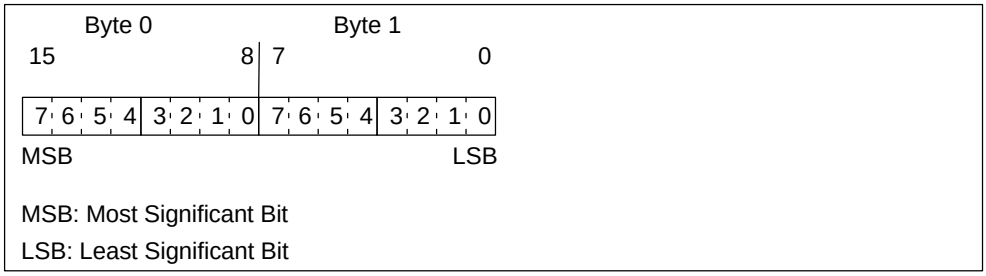

Figure A-9 Examples of the DATE Data Type

A.9 Format of the TIME_OF_DAY Data Type

A variable with TIME_OF_DAY (time of day) data type takes up a double word. It contains the number of milliseconds since the day commenced (0:00 hours) as an unsigned fixed-point number. The representation contains details for hours, minutes and seconds. Specification of the milliseconds is not necessary.

The TIME_OF_DAY data type is stored as an unsigned integer in milliseconds, where zero is equal to midnight.

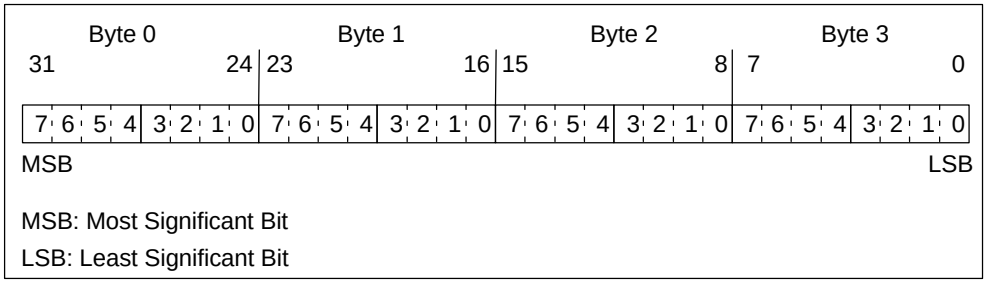

Figure A-10 Example of the TIME_OF_DAY Data Type

A.10 Format of the STRING Data Type

A string is a string of ASCII characters of any length. The maximum length is 254 characters. If a length is not specified, the default setting is 254 characters.

Example: STRING [55]: 'The character string can consist of up to 55 characters.'

The following example shows the byte order when specifying the data type STRING [4] with the output value 'AB'.

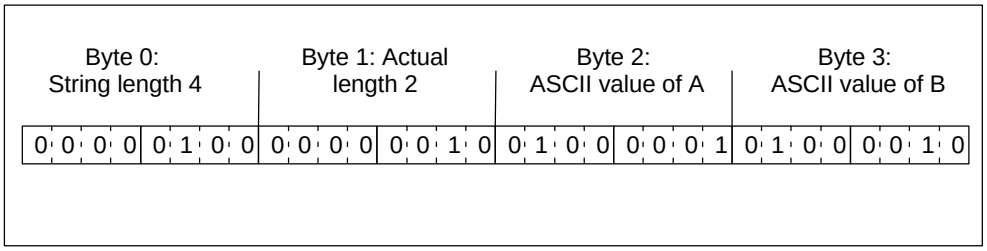

Figure A-11 Examples of the STRING Data Type

Glossary

P

PC data format

By PC data format we mean the “little endian” format (also known as the Intel format) as used within the PC, for example.

S

S7 data format

By S7 data format we mean the “big endian” format (also known as the Motorola format) as used within the world of the SIMATIC S7, for example.

Index

G

H

I

L

M

O

P

R

S

T

W