FDM RTX RELEASE 2
User Reference Manual

ALS 53330 c–en

First issue: 04–2000
This edition: 01–2001

Meaning of terms that may be used in this document / Notice to readers

WARNING

Warning notices are used to emphasize that hazardous voltages, currents, temperatures, or other conditions that could cause personal injury exist or may be associated with use of a particular equipment.

In situations where inattention could cause either personal injury or damage to equipment, a Warning notice is used.

Caution

Caution notices are used where there is a risk of damage to equipment for example.

Note

Notes merely call attention to information that is especially significant to understanding and operating the equipment.

Index letter	Date	Nature of revision
b	**08–2000**	• Chapter 4 functions *rtfdm_get_version*, *rtfdm_get_EOC_count* and *rtfdm_initialize_network*
c	**01–2001**	• Use of the CC141 board.

Revisions

1. PURPOSE OF MANUAL AND DOCUMENTED VERSION

This document is the manual for the implementation of FIP DEVICE MANAGER Version 4 designed for a Windows NT 4 environment on PCs equipped with VenturCom's RTX real time extension of the CC139, CC140 and CC141 PCI communication boards. This product therefore expands the FIP DEVICE MANAGER offer, within the framework of ALSTOM's WorldFIP technological offer, globally marketed under the FIPWARE trademark.

This document describes:

- installation, configuration and start–up,
- the special features of this version as regards the application interface, in relation to the basic interface offered by the FIP DEVICE MANAGER Version 4 function library.

This document in no way constitutes a training manual for the operating concepts and principles of the WorldFIP network.

Users, depending on whether they are installing this software on a PC to serve as a user application, or whether they plan to carry out the interfacing of an application, must have a working or advanced knowledge of WorldFIP operating principles.

Users who plan to develop the interfacing of an application with this software must follow the FIP DEVICE MANAGER Version 4 manual as it is indispensable to this task.

2. CONTENT OF THIS MANUAL

This user manual is structured in the following way:

Chapter 1: Introduction: General presentation of the software in relation to the hardware and software environment into which it is integrated.

Chapter 2: Installation: Commissioning of the software under a Windows environment on a PC equipped with a WorldFIP coupling board (installation, definition of parameters, start–up, stop).

Chapter 3: Use.

Chapter 4: Detailed description of the functions: Description of the programming interface.

Chapter 5: Error List.

Chapter 6: Miscellanous.

3. RELATED PUBLICATIONS

The documents quoted in this manual are specified in the text in square brackets and listed in this section:

[1] ALS 50278 FIP DEVICE MANAGER Software Version 4 User Reference Manual

[2] ALS 53316 WorldFIP–PCI Communication Board CC138/139/140/141 V1 User Reference Manual

For more information on WorldFIP and hardware components:

[3] ALS 50249 FIP Network General Introduction

[4] ALS 50262 FULLFIP2 User Reference Manual

4. WE WELCOME YOUR COMMENTS AND SUGGESTIONS

ALSTOM strives to produce quality technical documentation. Please take the time to fill in and return the ”Reader’s Comments” page if you have any remarks or suggestions

Reader's comments

ALS 53330 c–en FDM RTX RELEASE 2 User Reference Manual

Your main job is:

- ☐ System designer
- ☐ Distributor
- ☐ System integrator
- ☐ Installer
- ☐ Programmer
- ☐ Maintenance
- ☐ Operator
- ☐ Other (specify below)

If you would like a personal reply, please fill in your name and address below:

COMPANY: . . . . . NAME: . . . . . .

ADDRESS: . . . . . .

. . . . . COUNTRY: . . . . . .

Send this form directly to your ALSTOM sales representative or to this address:

ALSTOM Technology
Technical Documentation Department (TDD)
23–25, Avenue Morane Saulnier
92364 Meudon la Forêt Cedex
France
Fax: +33 (0)1 46 29 10 21

All comments will be considered by qualified personnel.

REMARKS

Continue on back if necessary.

Reader's comments

Contents

Contents

Figures

Tables

Chapter 1 Introduction

The FIP DEVICE MANAGER RTX Release 2 software package represents the adaptation of FDM V4 on VenturCom's RTX real time extension of Windows NT. This software package drives a PCI communication board:

- CC139: WorldFIP communication board for 1 Mbit/sec,
- CC140: WorldFIP communication board for 2.5 Mbits/sec,
- CC141: WorldFIP communication board for 5 Mbits/sec.

The FIP DEVICE MANAGER RTX Release 2 software package is delivered as a dynamic link library RTSS DLL.

The main files contained in the supply are:

- fdmrtdll.rtss: this is the executable module,
- fdmrtdll.lib: this is the import library file that must be linked with the user application.

The user process must be linked to the **fdmrtdll.lib** import library. The user process must be generated by a Visual C++ 6.0 compiler on RTX 4.3.1 or later.

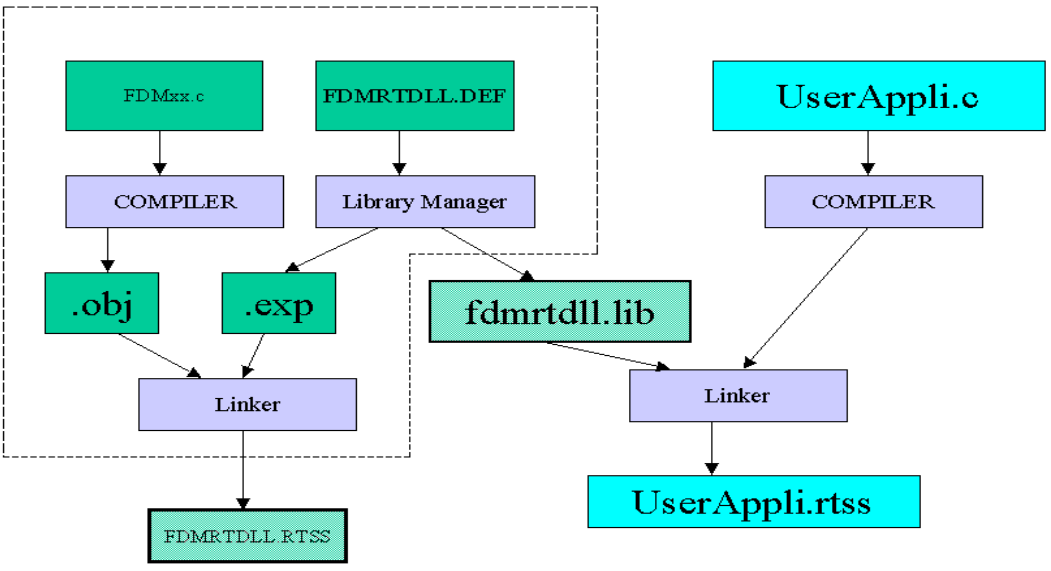


Figure 1.1 – Generation of an FDM–RTX application

Note

Each user RTSS process can be associated with only one board (that means if you have two boards you must develop two RTSS processes).

The FIP DEVICE MANAGER RTX Release 2 software package offers the user many functions that derive from the basic FDM V4 software package. The details and the description of the different concepts of WorldFIP communications are described in [1]. The main tasks implemented are the following:

ENSURING SYSTEM OPERATION

This functionality is dedicated to the management of components required to build WorldFIP connection points, i.e. FULLFIP2, medium redundancy circuits and private memories. It consists of the following basic functions:

- configure and manage one or N FULLFIP2 components,
- carry out the self–tests of the WorldFIP connection point(s) (optional),
- process events coming from the network(s),
- manage memory resources,
- develop the various time–outs required for other functions,
- monitor the quality of the media,
- prevent degradation,
- prevent latent faults (in double–medium),
- allow the subscriber to run as a FIPIO manager.

MANAGING AE/LE AND THE MPS VARIABLES

This functionality enables the application layer services to be performed which enables the user to have access to the MPS variables through unconnected logic entities called AE/LE. It consists of the following basic functions:

- create and delete an MPS AE/LE,
- start–up and stop an MPS AE/LE,
- add a variable in an MPS AE/LE,
- read a variable consumed in an AE/LE with or without dynamic refresh status,
- write a variable produced in an AE/LE with or without dynamic refresh status,
- manage supposed *pure sync* variables.

MANAGING THE BUS ARBITRATOR(S)

This functionality consists of the following basic functions:

- load a macrocycle,
- start or stop a macrocycle,
- change a macrocycle,
- resynchronize a macrocycle,
- manage election of a bus arbitrator.

HANDLING SM–MPS NETWORK MANAGEMENT VARIABLE

This functionality enables network management services to be performed and provides the user with access to the SM–MPS variables. It consists of the following basic functions:

- management of the ”list of equipment present” variables,
- management of the presence variable,
- management of the identification variable,
- management of the report variable,
- management of the BA sync variable.

MANAGING ”WORLDFIP DATA LINK LAYER” MESSAGES

This functionality enables the data connection layer services to be performed. These provide the user with access to messages at ”DLL WorldFIP” level. It consists of the following basic functions:

- configure messaging,
- transmit a message,
- receive a message.

MANAGING TIME

This functionality consists of the following basic functions:

- managing election of time variable producer,
- time variable read,
- time variable write.

Chapter 2 Installation

1. SUPPLY

The main files contained in the supply are:

- **fdmrtdll.rtss**: the executable module,
- **fdmrtdll.lib**: the import library file that must be linked to the user application,
- **messages.dll**: dll that processes errors,
- **Setup**: utility used for the installation,
- **ScanBus**: utility used for association of a board to a RTSS process
- **ErrorLookup**: utility that gets you the details of an error
- **header files**,
- **examples**,
- **documents**: User Reference Manual

2. INSTALLATION

Double click **Setup.exe** program.

3. START–UP/STOP

START

Several methods are available to start **fdmrtdll.rtss**:

- from the command prompt enter: **RTSSRUN fdmrtdll.rtss,**
- to be started at boot time, from the command prompt enter: **RTSSRUN /b fdmrtdll.rtss,**
- directly from a WIN32 process or a RTSS process: you must call the **CreateProcess** function that executes RTSSRUN.

To start the RTSS application you can use one of the above methods used to start the **fdmrtdll.rtss** dynamic link library.

Note

The **fdmrtdll.rtss** dynamic link library must be started before starting the user process**.**

STOP

From the command prompt enter: **RTSSKILL [/b] process_number.**

Chapter 3 Use

1. PROGRAMMING INTERFACE

The FIP DEVICE MANAGER RTX Release 2 software package offers the user many functions that derive from the basic FDM V4 software package.

The interface which is then available differs principally from the basic FDM V4 interface in terms of:

- management of initialisation functions (hardware, software, network, medium) and the management of communication (ticks, interrupts, etc.),
- services for the exchange of variables, messages and bus arbitration, which are limited to standard services,
- the time management service, which is limited to broadcasting the time of the PC and managing the redundancy of the time producer,
- additional functions, used to create the contexts necessary for RTX applications and manage error processing.

The **user RTSS process** must be linked to the **fdmrtdll.lib** import library. The user RTSS process must be generated by a Visual C++ 6.0 compiler on RTX 4.3.1. In addition, the RTSS must be compiled in ”struct member alignment: 8 bytes” mode, which is the default option proposed by the Visual C++ 6.0 compiler or later.

Applications can be created that partition their time–critical functions to run in the RTSS environment and non–time–critical functions to run in the Win32 environment. Processes in both environments can communicate and synchronise through the shared RTX Interprocess Communication (IPC) objects. Processes in the RTSS environment perform the high–speed deterministic control functions of the application. The process in the Win32 environment can utilise the complete range of Win32 functions as graphical displays and data storage.

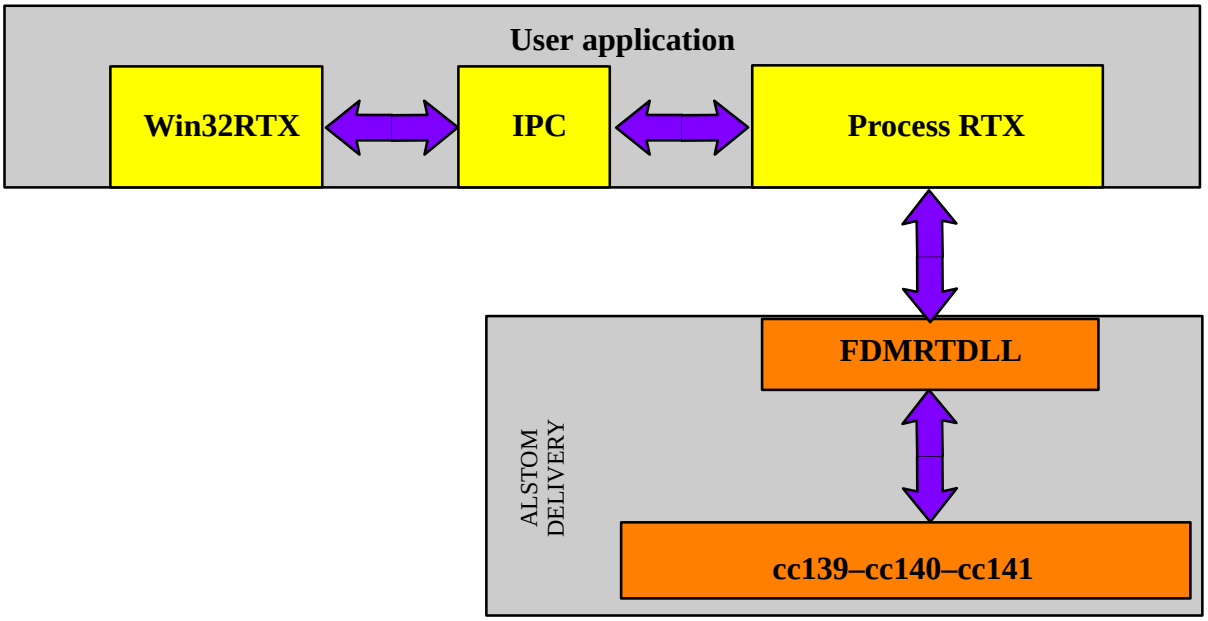


Figure 3.1 – Application Programming Interface

2. TIME MANAGEMENT

Depending on the settings made, the PC may choose to broadcast its time over the network thanks to the ID9802 variable, using specific FDM V4 mechanisms. The time is displayed in POSIX format.

The variable produced is updated automatically every 500 ms, with a guaranteed precision of 100 ns for the recopy operation. The time broadcast over the network uses a level of precision set by *Time_slot* (selected during configuration).

Note

Therefore, under general conditions of use at 1 Mbit/s, the difference between the time of the PC and the time broadcast will be less than 62.5 µs if *Time_slot* has been selected with this value.

Management of time producer election may be activated on initialisation and is used to select a time producer from amongst various devices that are capable of time production on the same network. Channel 1 must be configured before the election mechanism will work.

3. PERFORMANCE

Performance in terms of device exchange capacity has been defined as follows:

Maximum number of AELE:	40
Maximum number of macrocycles:	2
Maximum number of msg contexts:	40
Available messaging channels:	0 for aperiodic 1..8 for periodic
fdm_ticks_counter period:	100 ms

4. ERROR PROCESSING

The **GetLastError** function returns the calling thread's last–error code value. The last–error code is maintained on a per–thread basis. Multiple threads do not overwrite each other's last–error code.

DWORD GetLastError(VOID)

Parameters

This function has no parameters.

Return Values

The return value is the calling thread's last–error code value. Functions set this value by calling the SetLastError function. The **Return Value** section of each reference page notes the conditions under which the function sets the last–error code.

Chapter 4 Detailed description of the functions

1. INITIALISATION

- Use of only one board:

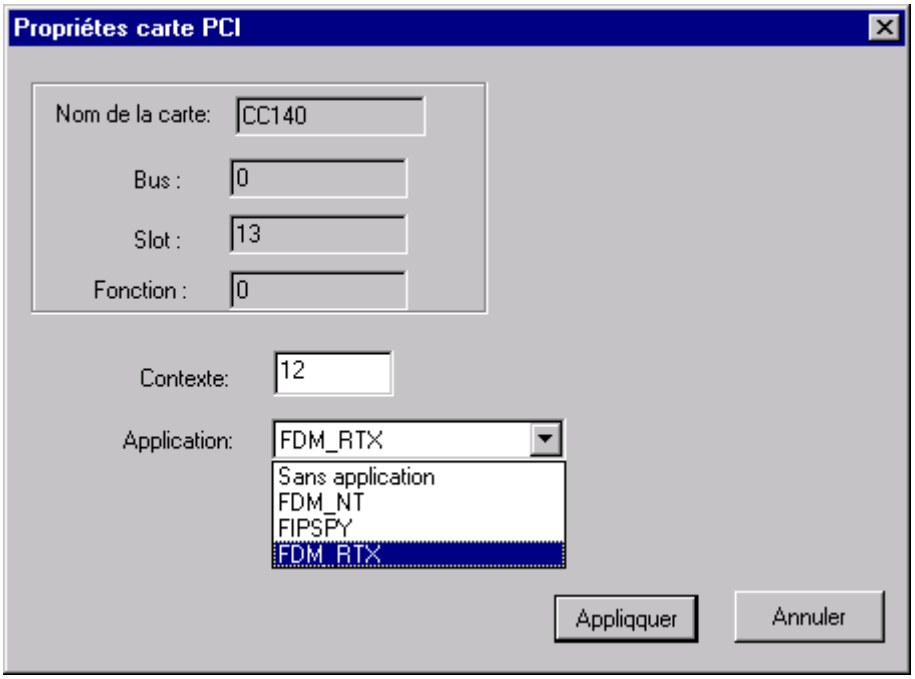
  In this case you must call the function **rtfdm_create_monoboard_context**.

- Use of more than one board:

  In this case you must call the function **rtfdm_create_multiboard_context**.

  Before calling this function you must detect the characteristics of your PCI board by using the **FipBoardConf.exe** utility.

  The following screen gives you the main parameters that you must enter in the **rtfdm_create_multiboard_context** function.



FDM_INSTANCE = rtfdm_create_multiboard_context (0, 13, 0,...);

1.1. rtfdm_create_multiboard_context

RTSS_FDM *rtfdm_create_multiboard_context (

ULONG *Bus,*
ULONG *Slot,*
ULONG *Funct,*
VOID (RTFCNDCL **ShutDownRoutine*)(PVOID *Context,* LONG *Reason*));

Constructor of a FDM–RXT instance. This function initialises all the registers of the board.

Parameters

Bus: number of PCI bus where the board is plugged

Slot: Slot number

Funct: function number

ShutDownRoutine: The handler function to call when **FDM–RTSS** delivers the stop notification.

Context: The pointer to the **RTSS_FDM** reference

Reason: The reason argument may have one of the following values according to the reason for the notification:

RT_SHUTDOWN_NT_SYSTEM_SHUTDOWN

The system is starting a normal shutdown. Shortly after all shutdown handlers have been executed, Windows NT will stop.

RT_SHUTDOWN_NT_STOP

Windows NT has stopped (i.e., blue screen). RTSS will continue to operate with service restrictions.

Return Values

A stop handler object has been correctly instantiated when a valid handle is returned. Otherwise, INVALID_HANDLE _VALUE is returned and you should call GetlastError for more detailed information.

1.2. rtfdm_create_monoboard_context

RTSS_FDM *rtfdm_create_monoboard_context (

VOID (RTFCNDCL **ShutDownRoutine*) (PVOID *Context,* LONG *Reason*));

Constructor of a FDM–RXT instance. This function scans one PCI board and initialises all the registers of this board.

Parameters

ShutDownRoutine: The handler function to call when FDM–RTSS delivers the stop notification.

Context: The pointer to the **RTSS_FDM** reference

***Reason*:** The reason argument may have one of the following values according to the reason for the notification:

RT_SHUTDOWN_NT_SYSTEM_SHUTDOWN

The system is starting a normal shutdown. Shortly after all shutdown handlers have been executed, Windows NT will stop.

RT_SHUTDOWN_NT_STOP

Windows NT has stopped (i.e., blue screen). RTSS will continue to operate with service restrictions.

Return Values

A stop handler object has been correctly instantiated when a valid handle is returned. Otherwise, INVALID_HANDLE_VALUE is returned and you should call GetlastError for more detailed information.

1.3. rtfdm_delete_context

void rtfdm_delete_context (RTSS_FDM *Ref);

Destructor of a FDM–RXT instance.

Parameters

Ref: reference of a RTX–FDM instance.

No Return Values

1.4. rtfdm_initialize

BOOL rtfdm_initialize(RTSS_FDM *Ref);

Initialization of RTX environment parameters.

Parameters

Ref: reference of a RTX–FDM instance.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

Note

The CC1xx hardware watchdog can be removed if you write 0 in Ref → Port Verif Altera before calling this function.

1.5. rtfdm_initialize_network

BOOL
rtfdm_initialize_network (

RTSS_FDM	***Ref,***
const FDM_CONFIGURATION_SOFT	***Soft,***
const FDM_CONFIGURATION_HARD	***Hard,***
const FDM_IDENTIFICATION	***Ident**

);

initialisation of an FDM instance

Parameters

Ref: reference of an RTX–FDM instance.

Soft:

```
typedef struct {
    ~~Ushort Type~~;
        // MESSAGE_RECEPTION_AUTHORIZED |
        // TWO_BUS_MODE |
        // TWO_IMAGE_MODE
        enum  _FULLFIP_Mode_List Mode;
        Ushort  TSlot;
        Ushort  NB_OF_USER_MPS_VARIABLE;        // 150
        Ushort  BA_Dim;                         // 0x2000
        ~~Ulong  FULLFIP_RAM_Dim;~~             // interne
        Ushort  NB_OF_DIFFERENT_ID_PROG_BA;     // 0x100
        Ushort Nr_Of_Repeat;                    // 1
        Ushort Nr_Of_Tx_Buffer[9];              // {4,4,4,4,4,4,4,4,4}
        void (*User_Present_List_Prog )
        ( struct _FDM_REF*,FDM_PRESENT_LIST * );
        Ushort (*User_Identification_Prog )
        ( struct _FDM_REF*, FDM_IDENT_VAR * );
```

```
        Ushort (*User_Report_Prog )
        ( struct _FDM_REF*, FDM_REPORT_VAR * );
        Ushort (*User_Presence_Prog )
        ( struct _FDM_REF*, FDM_PRESENCE_VAR * );
        void (*User_Synchro_BA_Prog )
        ( struct _FDM_REF*, FDM_SYNCHRO_BA_VAR * );
        Ushort          Test_Medium_Ticks;          // interne,
        Ushort          Time_Out_Ticks;             // * 100 ms
        Ushort          Time_Out_Msg_Ticks;         // * 100 ms
        Ushort          Online_Tests_Ticks;         // 0
        Ushort          Default_Medium_threshold;   // 2%
            struct {
            Ushort TIMER_CNT_REGISTER;
            Ushort MODE_REGISTER;
            } User_responsability;
        void (*User_Signal_Mps_Aper) ( struct _FDM_REF* );  // interne
        void (*User_Signal_Smmps ) ( struct _FDM_REF* );    // interne
        void (*User_Signal_Send_Msg) ( struct _FDM_REF* );  // interne
        void (*User_Signal_Rec_Msg ) ( struct _FDM_REF* );  // interne
        void * User_Ctxt;                               // interne
    } FDM_CONFIGURATION_SOFT;
```

Hard:

```
    typedef struct _FDM_CONFIGURATION_HARD {
        Uchar           K_PHYADR;
        char            MySegment;
        Uchar           Reserved[2];
        __Port_Type__ LOC_FIP[8];                   // interne
        __Port_Type__ LOC_FIPDRIVE[4];              // interne
        Ushort volatile * FREE_ACCES_ADDRESS;       // interne
        void (*User_Reset_Component)                // interne
        ( struct _FDM_CONFIGURATION_HARD * );
        void (*User_Signal_Fatal_Error)             // interne
        ( struct _FDM_REF * Ref,FDM_ERROR_CODE );
        void (*User_Signal_Warning )                // interne
        ( struct _FDM_REF * Ref,FDM_ERROR_CODE );
```

```
        MEMORY_RN *Memory_Management_Ref;        // interne
        struct _FDM_REF *Ptr_Autotests;          // interne
    } FDM_CONFIGURATION_HARD;
```

Ident:

```
    typedef struct {
        char * Vendor_Name;
        char * Model_Name;
        char * Revision;
        char * Tag_Name;                 // interne
        char * SM_MPS_Conform;           // interne
        char * SMS_Conform;              // interne
        char * PMDP_Conform;             // interne
        char * Vendor_Field;
    } FDM_IDENTIFICATION;
```

Return Values

The function returns TRUE if it completes successfully, otherwise it returns FALSE. If FALSE is returned, then you should call GetlastError for more detailed information.

1.6. rtfdm_valid_medium

BOOL rtfdm_valid_medium (RTSS_FDM * Ref, enum _MEDIUM_DEF Medium);

Choice of media to be used.

Parameters

Ref: reference of a RTX–FDM instance.

Medium: medium

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

1.7. rtfdm_stop_network

BOOL rtfdm_stop_network (RTSS_FDM * Ref);

Stop the network that has been started by the function **rtfdm_initialize_network.**

Parameters

Ref: reference of a RTX–FDM instance.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

2. AE/LE AND MPS VARIABLES MANAGEMENT

2.1. rtfdm_ae_le_create

BOOL rtfdm_ae_le_create (

RTSS_FDM * Ref,

int AE_LE_RANG,

int AE_LE_DIM);

Creation of an AE_LE.

Parameters

Ref: reference of a RTX–FDM instance.

AE_LE_RANG: row of the AE_LE. Value [0..47].

AE_LE_DIM: maximum number of variables which may make up this AE_LE.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

2.2. rtfdm_ae_le_delete

BOOL rtfdm_ae_le_delete (

RTSS_FDM * Ref,

int AE_LE_RANG);

Deletion/destruction of an AE_LE.

Parameters

Ref: reference of a RTX–FDM instance.

AE_LE_RANG: row of the AE_LE. Value [0..47].

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

2.3. rtfdm_ae_le_start

BOOL rtfdm_ae_le_start (

RTSS_FDM * Ref,

int AE_LE_RANG);

Start–up of an AE_LE.

Parameters

Ref: reference of a RTX–FDM instance.

AE_LE_RANG: row of the AE_LE. Value [0..47].

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

2.4. rtfdm_ae_le_stop

BOOL rtfdm_ae_le_stop (

RTSS_FDM * Ref,

int AE_LE_RANG);

Stopping an AE_LE

Parameters

Ref: reference of a RTX–FDM instance.

AE_LE_RANG: row of the AE_LE. Value [0..47].

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

2.5. rtfdm_mps_var_create

BOOL rtfdm_mps_var_create (

RTSS_FDM * Ref,

int AE_LE_RANG,

const FDM_XAE * Var_Param);

Defines the parameters of a variable in an AE_LE.

Parameters

Ref: reference of a RTX–FDM instance.

AE_LE_RANG: row of the AE_LE. Value [0..47].

Var_Param: pointer to a structure describing the parameters of the variable (Refer to [1] for the description of this structure).

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

2.6. rtfdm_mps_var_write_loc

BOOL rtfdm_mps_var_write_loc (

RTSS_FDM * Ref,

int AE_LE_RANG,

int VAR_RANG,

USER_BUFFER_TO_READ *Data_Buffer);

Writing of an MPS variable.

Parameters

Ref: reference of a RTX–FDM instance.

AE_LE_RANG: row of the AE_LE. Value [0..9].

VAR_RANG: row of the variable in the AE_LE.

Data_Buffer: pointer to a USER_BUFFER_TO_READ type structure containing the data to be written. Refer to [1] for the description of this structure.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

2.7. rtfdm_mps_var_read_loc

BOOL rtfdm_mps_var_read_loc (

RTSS_FDM * Ref,

int AE_LE_RANG,

int VAR_RANG,

USER_BUFFER_TO_READ *Data_Buffer);

Reading of an MPS variable.

Parameters:

Ref: reference of a RTX–FDM instance.

AE_LE_RANG: row of the AE_LE. Value [0..9].

VAR_RANG: row of the variable in the AE_LE.

Data_Buffer: pointer to a USER_BUFFER_TO_READ type structure containing the data to be read. Refer to [1] for the description of this structure.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

2.8. rtfdm_mps_var_write_universal

BOOL rtfdm_mps_var_write_universal (

RTSS_FDM * Ref,

int AE_LE_RANG,

int VAR_RANG,

USER_BUFFER_TO_READ *Data_Buffer);

This function makes it possible to write an MPS variable whether it is local or remote. If the variable is remote then an aperiodic request is made. When the variable is sent then a user callback procedure is activated (is User_Signal_Var_Prod field of FDM_XAE structure – refer to [1] for more information).

Parameters

Ref: reference of a RTX–FDM instance.

AE_LE_RANG: row of the AE_LE. Value [0..9].

VAR_RANG: row of the variable in the AE_LE.

Data_Buffer: pointer to a USER_BUFFER_TO_READ type structure containing the data to be written. Refer to [1] for the description of this structure.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

2.9. rtfdm_mps_var_read_universal

BOOL rtfdm_mps_var_read_universal(

RTSS_FDM * Ref,

int AE_LE_RANG,

int VAR_RANG);

This function makes it possible to read an MPS variable whether it is local or remote. If the variable is remote then an aperiodic request is made. When the variable is received then a user callback procedure is activated (is User_Signal_Var_Cons field of FDM_XAE structure – refer to [1] for more information).

Parameters

Ref: reference of a RTX–FDM instance.

AE_LE_RANG: row of the AE_LE. Value [0..9].

VAR_RANG: row of the variable in the AE_LE.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

3. SYNCHRONISATION

You can synchronise your application on the FIP synchronisation variable. You must proceed as follows:

1. Open the RTX event **FIP_EOC_Event**:
   HANDLE EvtEoc = **RtOpenEvent**(0,FALSE,L"**FIP_EOC_EventXX**"); where XX is the slot number (hexadecimal) given by the ScanBus utility;

2. Wait on this Event
   RtWaitForSingleObject (EvtEoc, INFINITE);

Note

The event **FIP_EOC_Event** is created by the **fdmrtdll.rtss** library with the following parameters:

- InitialState = FALSE,
- Automatic reset mode.

4. BUS ARBITRATOR MANAGEMENT

4.1. rtfdm_ba_load_macrocycle_fipconfb

BOOL rtfdm_ba_load_macrocycle_fipconfb (

RTSS_FDM * Ref,

int BA_RANG,

const Ushort* ba_fipconfb);

Used to create and load a macrocycle.

Parameters

Ref: reference of a RTX–FDM instance.

BA_RANG: row of the macrocycle [0 .. 1].

ba_fipconfb: pointer to a table generated in the FIPCONFB tool format. Refer to [1].

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

4.2. rtfdm_ba_load_macrocycle_manual

BOOL rtdm_ba_load_macrocycle_manual (

RTSS_FDM * Ref,

int BA_RANG,

int Nb_of_List,

int Nb_of_Instruction,

unsigned short Label,

const PTR_LISTS * List,

const PTR_INSTRUCTIONS * Instructions);

Used to create and load a macrocycle.

Parameters

Ref: reference of a RTX–FDM instance.

BA_RANG: row of the macrocycle [0 .. 1].

Nb_of_List Number of lists described.;

Nb_of_Instruction: Number of instructions described.

Label: Internal macrocycle label.

Lists: Pointer to a data structure of the PTR_LISTS type, described above, containing the sequences of identifiers to be circulated.

Instructions: Pointer to a data structure of the PTR_INSTRUCTIONS type, described above, containing the sequences of instructions to be executed.

Refer to **[1]** for more information.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

4.3. rtfdm_ba_delete_macrocycle

BOOL rtfdm_ba_delete_macrocycle (

RTSS_FDM * Ref,

int BA_RANG);

Used to delete a macrocycle.

Parameters

Ref: reference of a RTX–FDM instance.

BA_RANG: row of the macrocycle [0 .. 1].

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

4.4. rtfdm_ba_status

BOOL rtfdm_ba_status (

RTSS_FDM * Ref,

BA_INF_STATUS Resultats);

Used to read the status of the BA function of the station.

Parameters

Ref: reference of a RTX–FDM instance.

Resultats: pointer to a BA_INF_STATUS type structure described below and containing the status of the BA. Refer to [1] for more information.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

4.5. rtfdm_ba_start

BOOL rtfdm_ba_start (

RTSS_FDM * Ref,

int BA_RANG);

Used to start up a macrocycle.

Parameters

Ref: reference of a RTX–FDM instance.

BA_RANG: row of the macrocycle [0 .. 1].

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

4.6. rtfdm_ba_stop

BOOL rtfdm_ba_stop (RTSS_FDM * Ref);

Used to stop the current macrocycle.

Parameters

Ref: reference of a RTX–FDM instance.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

4.7. rtfdm_ba_set_parameters

BOOL rtfdm_ba_set_parameters (

RTSS_FDM * Ref

enum _BA_SET_MODE BA_Mode,

unsigned char MAX_Subscriber,

unsigned char MAX_Priority);

Used to modify the parameters used in the time delay calculations of the BA function of the station.

Parameters

Ref: reference of a RTX–FDM instance.

BA_Mode: Refer to [1].

MAX_Subscriber: Refer to [1].

MAX_Priority: Refer to [1].

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

4.8. rtfdm_ba_set_priority

BOOL rtfdm_ba_set_parameters (

RTSS_FDM * Ref

unsigned char *Priority_Level);*

Used to modify the parameters used in the time delay calculations of the BA function of the station.

Parameters

Ref: reference of a RTX–FDM instance.

Priority_Level: desired level of priority [0…15], with 0 being the highest priority.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

5. MANAGING WorldFIP DATA LINK LAYER MESSAGES

5.1. rtfdm_channel_create

BOOL rtfdm_channel_create (

RTSS_FDM * Ref,

FDM_CHANNEL_PARAM *Channel_param);

Used to create a periodic messaging channel. Eight periodic messaging channels numbered from 1 to 8 can be created.

Parameters:

Ref: reference of a RTX–FDM instance.

Channel_param: pointer to a FDM_CHANNEL_PARAM type Structure. Refer to [1].

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

5.2. rtfdm_channel_delete

BOOL rtfdm_channel_delete (

RTSS_FDM * Ref,

Ushort *Channel_nr);

Used to delete a periodic messaging channel.

Parameters:

Ref: reference of a RTX–FDM instance.

Channel_nr: number of the channel to be deleted

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

5.3. rtfdm_messaging_fullduplex_create

BOOL rtfdm_messaging_fullduplex_create (

RTSS_FDM * Ref,

int Rang,

FDM_MESSAGING_FULLDUPLEX * Data_Buffer);

Used to create a new FULLDUPLEX messaging system context.

Parameters:

Ref: reference of a RTX–FDM instance.

Rang: messaging context number.

Data_Buffer: pointer to an FDM_MESSAGING_FULLDUPLEX type data structure describing the context. Refer to [1].

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

5.4. rtfdm_messaging_to_send_create

BOOL rtfdm_messaging_to_send_create (

RTSS_FDM * ***Ref*****,**

int ***Rang*****,**

FDM_MESSAGING_TO_SEND * Data_Buffer);

Used to create a messaging context for sending.

Parameters:

Ref: reference of a RTX–FDM instance.

Rang: messaging context number.

Data_Buffer: pointer to an FDM_MESSAGING_TO_SEND type data structure describing the context. Refer to [1].

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

5.5. rtfdm_messaging_to_rec_create

BOOL rtfdm_messaging_to_rec_create (

RTSS_FDM * Ref,

int Rang,

FDM_MESSAGING_TO_REC * Data_Buffer);

Used to create a messaging context for receiving.

Parameters:

Ref: reference of a RTX–FDM instance.

Rang: messaging context number.

Data_Buffer: pointer to an FDM_MESSAGING_TO_REC type data structure describing the context. Refer to [1].

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

5.6. rtfdm_messaging_delete

BOOL rtfdm_messaging_to_rec_create (

RTSS_FDM * Ref,

int Rang);

Used to delete a messaging context.

Parameters:

Ref: reference of a RTX–FDM instance.

Rang: messaging context number.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

5.7. rtfdm_send_message

BOOL rtfdm_send_message (

RTSS_FDM * Ref,

int Rang,

FDM_MSG_TO_SEND * Data_Buffer);

Used to request the transmission of a message.

Parameters:

Ref: reference of a RTX–FDM instance.

Rang: messaging context number.

Data_Buffer: pointer to an FDM_MSG_TO_SEND type data structure. Refer to [1].

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

5.8. rtfdm_msg_ref_buffer_free, rtfdm_msg_data_buffer_free

BOOL rtfdm_msg_data_buffer_free (FDM_MSG_R_DESC *Memory);

BOOL rtfdm_msg_ref_buffer_free (FDM_MSG_RECEIVED *Memory);

Used to free the memory zone which contains a received message or received message descriptor.

Parameters:

Memory: pointer to the data structure to be released. Refer to [1].

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

6. MANAGING SM–MPS NETWORK MANAGEMENT VARIABLES

6.1. rtfdm_read_report

BOOL rtfdm_read_report (

RTSS_FDM * Ref,

int subscriber);

Used to read a subscriber report variable. Refer to [1].

Parameters

Ref: reference of a RTX–FDM instance.

subscriber: physical address of the subscriber whose report variable is to be read.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

6.2. rtfdm_read_present_list

BOOL rtfdm_read_present_list (

RTSS_FDM * Ref);

Used to read a subscriber present list* variable. Refer to [1].

Parameters

Ref: reference of a RTX–FDM instance.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

6.3. rtfdm_read_identification

BOOL rtfdm_read_identification (

RTSS_FDM * Ref,

int subscriber);

Used to read the subscriber identification variable. Refer to [1].

Parameters

Ref: reference of a RTX–FDM instance.

subscriber: physical address of the subscriber whose identification variable is to be read.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

6.4. rtfdm_read_presence

BOOL rtfdm_read_presence (

RTSS_FDM * Ref,

int subscriber);

Used to read the subscriber presence variable. Refer to [1].

Parameters

Ref: reference of a RTX–FDM instance.

subscriber: physical address of the subscriber whose presence variable is to be read.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

6.5. rtfdm_read_ba_synchronize

BOOL rtfdm_read_ba_synchronize (

RTSS_FDM * Ref);

Used to read the network BA sync variable. Refer to [1].

Parameters

Ref: reference of a RTX–FDM instance.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

7. MANAGING MPS TIME VARIABLE

Refer to [1] to see the format of the MPS time variable.

Notes

1. The RTSS timer CLOCK1 will be used by FDM after you have called the function **rtfdm_generic_time_initialize**.
2. When a station produces the MPS time variable CLOCK1 the value is sent.
3. When a station consumes the MPS time variable CLOCK1 the value is updated.
4. The **FdmRtxTime** example copies the RTSS CLOCK1 into the PC system clock.

7.1. rtfdm_generic_time_initialize

BOOL rtfdm_generic_time_initialize (

RTSS_FDM * Ref

const FDM_GENERIC_TIME_DEFINITION *User_param);

Creation of the MPS time variable, the messaging LSAPs required for the time producer redundancy management protocol to be operated and the possible start up of this protocol.

Parameters

Ref: reference of a RTX–FDM instance.

User_param: Refer to [1] for the details of this structure.

```
typedef struct {
enum BOOLEAN With_Choice_Producer;
enum BOOLEAN With_MPS_Var_Produced;
enum BOOLEAN With_MPS_Var_Consumed;
enum _FDM_MSG_IMAGE Image;
unsigned long Refreshment;                  // 2.5 Mhz: =400ms;
unsigned long Promptness;                   // 1 Mhz: =1000ms,
const unsigned long *Delta_Time_Location;
void (*User_Signal_Mode) (int Sens);
unsigned short Ticks_Election;              // 2.5 Mhz: =400ms;   // 1 Mhz: =1000ms,
unsigned short Channel_Nr;
} FDM_GENERIC_TIME_DEFINITION;
```

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

7.2. rtfdm_generic_time_set_priority

BOOL rtfdm_generic_time_set_priority (

RTSS_FDM * Ref

GT_PRIORITY priority);

Used to modify the subscriber priority that is involved in the election process of the MPS time variable producer.

Parameters

Ref: reference of a RTX–FDM instance.

priority: value of the priority to set [0 …15]; 0 is the highest priority

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

7.3. rtfdm_generic_time_set_candidate_for_election

BOOL rtfdm_generic_time_set_candidate_for_election(

RTSS_FDM * Ref,

enum FDM_BOOLEAN state);

This function enables the user to include or to exclude the subscriber in the procedure for managing redundancy of the MPS time variable producers (ID 9802).

Parameters

Ref: reference of a RTX–FDM instance.

state: FDM_TRUE to include the subscriber in the procedure for managing redundancy of the MPS time variable producers, or FDM_FALSE to exclude.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

7.4. rtfdm_generic_time_get_election_status

BOOL rtfdm_generic_time_get_election_status (

RTSS_FDM * Ref

EtatAbonnes *result);

Used to know the status of the subscriber regarding the election process of the time variable.

Parameters

Ref: reference of a RTX–FDM instance.

result: pointer to a EtatAbonnes type data structure, described below, that contains the status type def struct{

unsigned	abonnes: 8;
enum subscriber_states	state: 4;
unsigned	Prio: 4;

} EtatAbonnes;

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

7.5. rtfdm_generic_time_delete

BOOL rtfdm_generic_time_delete (

RTSS_FDM * Ref);

Deletion of the time management object.

Parameters

Ref: reference of a RTX–FDM instance.

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

8. UTILITIES

8.1. rtfdm_get_image

BOOL rtfdm_get_image (

RTSS_FDM * Ref,

FDM_IMAGE_NR *Value);

This function is used to find out which image the AE_LE and the messaging context are on.

Parameters

Ref: reference of a RTX–FDM instance.

Value: the result is IMAGE_1 or IMAGE_2

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

8.2. rtfdm_switch_image

BOOL rtfdm_switch_image (

RTSS_FDM * Ref,

enum IMAGE_NR image);

This function is used to request switching of the AE_LE and messaging context image.

Parameters

Ref: reference of a RTX–FDM instance.

image: IMAGE_1 or IMAGE_2

Return Values

The function returns TRUE if it is successfully completed, or FALSE otherwise. If FALSE is returned, you should call GetlastError for more detailed information.

8.3. rtfdm_get_version

rtfdm_get_version (

FDM_RTX_VERSION * Ptr);

This function is used to get the version of RTX DLL, FDM and FIPCODE.

Parameters

Ptr: pointer to a FDM_RTX_VERSION structure (see below).

typedef struct {

FDM_VERSION–ELEMENT fdm Rtx;

FDM_VERSION–ELEMENT fdm V4;

FDM_VERSION–ELEMENT fdm V6;

FDM_VERSION–ELEMENT;

}

Return Values

None

8.4. rtfdm_get_EOC_count

rtfdm_get_EOC_count (

RTSS_FDM *Ref,

short *Count

);

Used to get EOC Count.

Parameters

Ref: reference of a RTX–FDM instance.

Count: number of EOC inerrupts

Return Values

The function returns TRUE if it completes successfully or it returns FALSE otherwise. If FALSE is returned, then you should call GetlastError for more detailed information.

Chapter 5 Error List

There are 4 types of errors:

- SYSTEM : 0xc00000xx
- RTX : 0x000000xx
- FDM : 0x80040xxx – Refer to [1]
- DLL FDM : 0x8004exxx (see below)

Error Type	Value
AE_LE rank already defined	0x8004e300
AE_LE rank > Max	0x8004e301
AE_LE rank not defined	0x8004e302
Variable not defined	0x8004e303
Variable rank > max	0x8004e304
BA rank > 2	0x8004e305
BA not defined	0x8004e306
FDM not defined	0x8004e307
Allocate memory Fault	0x8004e308
MSG CTX > max	0x8004e309
MSG CTX rank already defined	0x8004e30a
MSG CTX rank not defined	0x8004e30b
Configuration Error	0x8004e30c
MPS time var. already defined	0x8004e30d

Table 5.1 – Errors types

The **ErrorLookup** utility shows you the details of the errors.

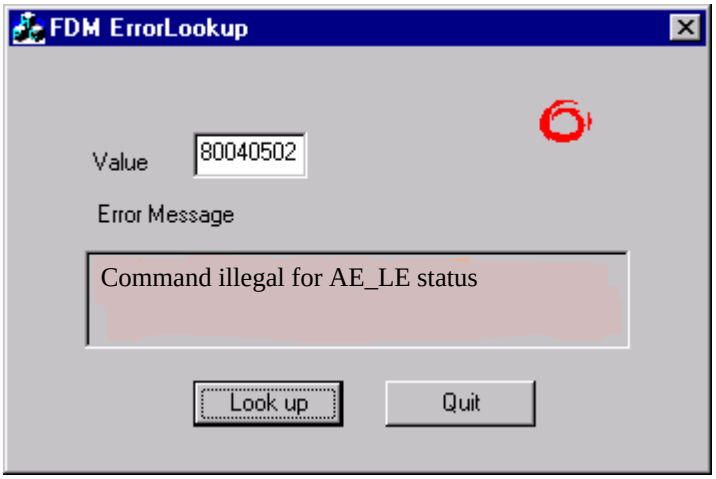


Figure 5.1 – FDM ErrorLookup Screen

Chapter 6 Miscellanous

To simplify the creation of an application, certain memory management functions (not documented, but used in the example program) can be used:

- _NEW_USER: creation of a memory pool,
- _DELETE_USER: deletion of a memory pool,
- _ALLOCATE_MEMORY: allocation of a memory bloc inside a memory pool,
- _FREE_MEMORY: freeing of a memory bloc inside a memory pool.