


Reference Manual AnyBus-S API

Rev. 1.10

HMS Industrial Networks AB


Germany +49- 721 - 96472 - 0
Japan +81- 45 - 478 -5340
Sweden +46- 35 - 17 29 20
U.S.A +1- 773 - 404 - 2271


sales-ge@hms-networks.com
sales-jp@hms-networks.com
sales@hms-networks.com
sales-us@hms-networks.com



Table of Contents

| | | |
|------------------|--|------|
| Preface | About This Manual | |
| | How To Use This Manual | P-1 |
| | Important User Information | P-1 |
| | Related Documentation | P-1 |
| | Revision List..... | P-1 |
| | Conventions Used In This Manual..... | P-2 |
| | Support | P-2 |
| Chapter 1 | Introduction to the AnyBus-S API | |
| | Features..... | 1-1 |
| | System Requirements..... | 1-1 |
| | Functional Overview | 1-2 |
| | Installation..... | 1-3 |
| Chapter 2 | AnyBus-S Overview | |
| | Dual Port Memory Contents (Shared Memory Area) | 2-1 |
| | Initialisation Process | 2-2 |
| Chapter 3 | Basic Operation | |
| | General..... | 3-1 |
| | Path Management..... | 3-2 |
| | <i>Path Related Functions</i> | 3-2 |
| | <i>Example</i> | 3-3 |
| Chapter 4 | Functions | |
| | Path Management..... | 4-2 |
| | <i>ABS_UserSelectPath()</i> | 4-2 |
| | <i>ABS_SelectPath()</i> | 4-4 |
| | <i>ABS_PathName()</i> | 4-5 |
| | <i>ABS_DestroyPath()</i> | 4-6 |
| | Mailbox Functions | 4-7 |
| | <i>ABS_SendMail()</i> | 4-7 |
| | <i>ABS_ReceiveMail()</i> | 4-8 |
| | Read/Write Operations..... | 4-9 |
| | <i>ABS_ReadControlArea()</i> | 4-9 |
| | <i>ABS_WriteContoLArea()</i> | 4-10 |
| | <i>ABS_ReadFieldbusArea()</i> | 4-11 |
| | <i>ABS_WriteFieldbusArea()</i> | 4-12 |
| | <i>ABS_ReadInArea()</i> | 4-13 |
| | <i>ABS_WriteInArea()</i> | 4-14 |
| | <i>ABS_ReadOutArea()</i> | 4-15 |

| | |
|---|------|
| Miscellaneous | 4-16 |
| <i>ABS_ModuleInfo()</i> | 4-16 |
| <i>ABS_ModuleStatus()</i> | 4-17 |
| <i>ABS_SetDpramSize()</i> | 4-18 |
| <i>ABS_Reset()</i> | 4-19 |
| <i>ABS_SetApplicationLed()</i> | 4-20 |
| | |
| Chapter 5 Data Structures and Enumerations | |
| <i>ABS_MailboxType</i> | 5-1 |
| <i>ABS_ModuleInfoType</i> | 5-3 |
| <i>ABS_ModuleStatusType</i> | 5-4 |
| <i>ABS_StatusType</i> | 5-5 |
| | |
| Chapter 6 Defines | |
| <i>ABS_AREA_XXX</i> | 6-1 |
| <i>ABS_FB_XXX</i> | 6-2 |
| <i>ABS_LED_XXX</i> | 6-4 |
| <i>ABS_MB_XXX</i> | 6-5 |
| <i>ABS_MODULE_TYPE_XXX</i> | 6-8 |
| <i>ABS_MODULE_STATUS_XXX</i> | 6-9 |
| <i>ABS_RESET_XXX</i> | 6-10 |
| | |
| Appendix A Advanced Functions | |
| Advanced Read/Write Operations | A-2 |
| <i>ABS_ParallelRead()</i> | A-2 |
| <i>ABS_ParallelWrite()</i> | A-3 |
| <i>ABS_ParallelVerifyRead()</i> | A-4 |
| <i>ABS_ParallelVerifyWrite()</i> | A-5 |
| Handshaking | A-6 |
| <i>ABS_RequestArea()</i> | A-6 |
| <i>ABS_ReleaseArea()</i> | A-7 |
| | |
| Appendix B Troubleshooting | |

About This Manual

How To Use This Manual

This document describes the functions of the AnyBus-S API. The reader of this document is expected to be familiar with the AnyBus-S platform, and software design in general.

This document is intended to be used in conjunction with the AnyBus-S Parallel Design Guide and the appropriate fieldbus appendix. (See ‘Related Documentation’ below).

Important User Information

The data and illustrations found in this document are not binding. We, HMS Industrial Networks AB, reserve the right to modify our products in line with our policy of continuous product development. The information in this document is subject to change without notice and should not be considered as a commitment by HMS Industrial Networks AB. HMS Industrial Networks AB assumes no responsibility for any errors that may appear in this document.

There are many applications of this product. Those responsible for the use of this device must ensure that all the necessary steps have been taken to verify that the application meets all performance and safety requirements including any applicable laws, regulations, codes, and standards.

AnyBus® is a registered trademark of HMS Industrial Networks AB. All other trademarks are the property of their respective holders.

The code examples in this document are written in C++, and are provided “as is”. The code has not been compiled nor tested, and is included for explanatory purposes only.

Related Documentation

| Document | Author |
|--------------------------------|--------|
| AnyBus-S Parallel Design Guide | HMS |
| AnyBus-M Parallel Design Guide | HMS |
| | |
| | |

Revision List

| Revision | Date | Author | Chapter(s) | Description |
|----------|------------|--------|------------|--|
| 0.10 | 2003-08-22 | PeP | All | Created |
| 0.20 | 2003-09-05 | PeP | All | Draft |
| 1.00 | 2003-09-14 | PeP | All | First release |
| 1.01 | 2003-09-29 | PeP | All | Minor corrections and adjustments |
| 1.10 | 2003-09-30 | PeP | All | Added AnyBus-S introduction & Troubleshooting. Various minor changes. |
| | | | | |

Conventions Used In This Manual

The following conventions are used throughout this manual:

- Numbered lists provide sequential steps
- Bulleted lists provide information, not procedural steps
- Hexadecimal values are written in the format NNNNh where NNNN is the hexadecimal value.
- Binary values are written in the format NNNNb, where NNNN is the binary value.
- The term ‘communication module’ is used when referring to the AnyBus communication module associated with a transport path.
- The term ‘application’ is used when referring the software application that uses the AnyBus-S API
- When using an AnyBus-S Master (AnyBus-M) the input / output definition is reversed. For more information, consult the AnyBus-M Design Guide.

Support

Europe (Sweden)

E-mail: support@hms-networks.com
Phone: +46 (0) 35 - 17 29 20
Fax: +46 (0) 35 - 17 29 09
Online: www.hms-networks.com

HMS America

E-mail: us-support@hms-networks.com
Phone: +1-773-404-2271
Toll Free: 888-8-AnyBus
Fax: +1-773-404-1797
Online: www.hms-networks.com

HMS Germany

E-mail: ge-support@hms-networks.com
Phone: +49-721-96472-0
Fax: +49-721-964-7210
Online: www.hms-networks.com

HMS Japan

E-mail: jp-support@hms-networks.com
Phone: +81-45-478-5340
Fax: +81-45-476-0315
Online: www.hms-networks.com

Introduction to the AnyBus-S API

The AnyBus-S API provides a means of writing portable, interface-independent applications that utilizes AnyBus hardware. The physical interface remains transparent to the application, meaning that the application can be made compatible with future interfaces by HMS with few or no adjustments.

The API also simplifies development drastically by taking care of low level functions such as handshaking etc., and providing easy to use functions for common AnyBus related tasks such as mailbox handling etc.

The API itself is compatible with all available AnyBus-S master and slave modules. Naturally, the AnyBus module must be supported by the physical interface used.

Features

- **Easy to use function calls for common AnyBus related tasks**
- **Ready for future interfaces**
- **Simultaneous access to multiple interfaces, regardless of interface type**
- **Portable**

System Requirements

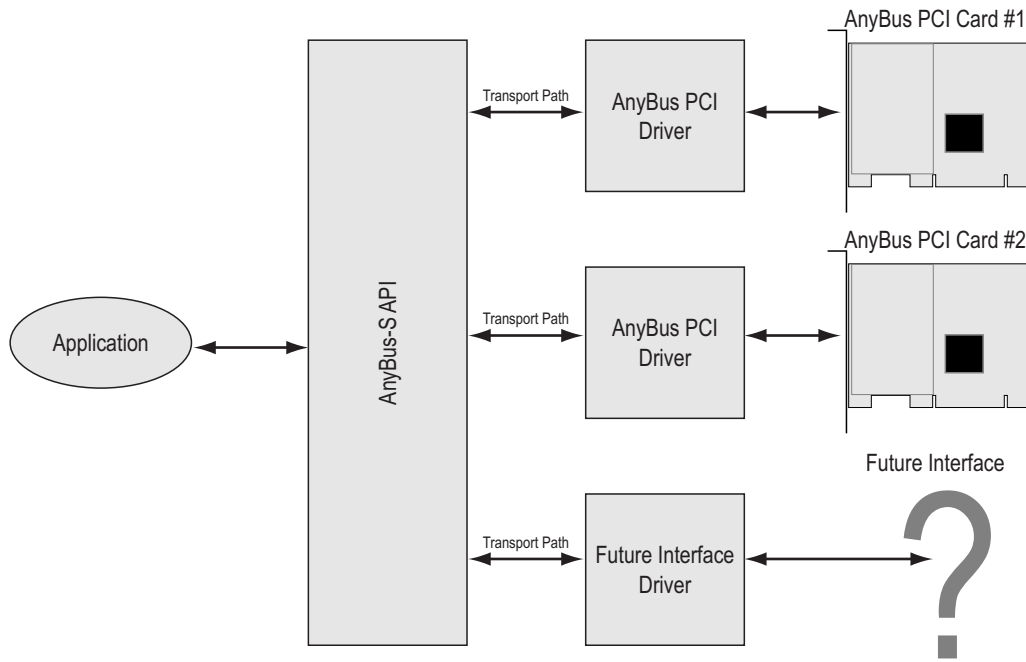
- **Supported Interface**
 - AnyBus PCI
- **Platforms**

Currently available for PC (x86) systems
- **Operating Systems**

Microsoft Windows 98/ME/NT/XP/2000
- **Supported Compilers / Languages**
 - Borland C++
 - Borland Delphi
 - Visual C++
 - Visual Basic

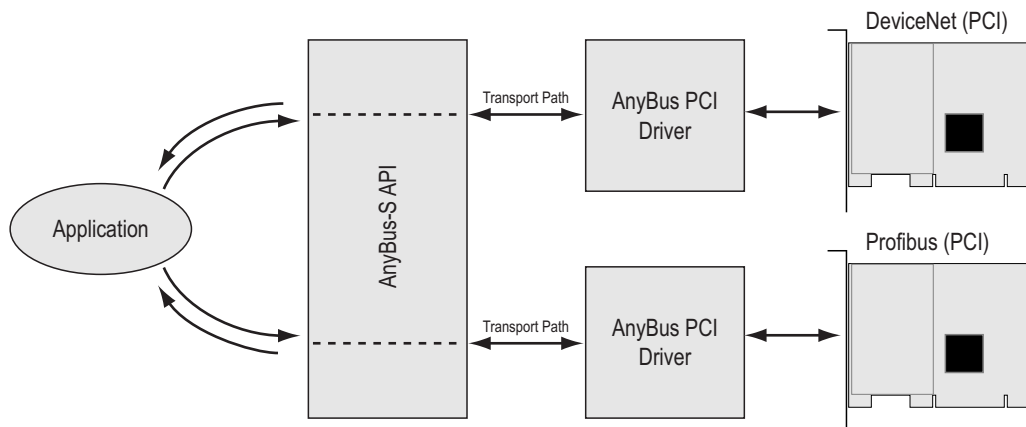
Functional Overview

The purpose of this API is to provide a function library that will aid the end user when designing code that interacts with an AnyBus communication module. The implementation does not use a direct memory architecture towards the AnyBus module as an embedded application would. Instead, an AnyBus specific hardware abstraction layer provides easy to use functions for common AnyBus related tasks.



As illustrated above, even if the physical interface and its' underlying driver is different, the software interface towards the application will always remain the same.

Multiple interfaces can be accessed simultaneously by the application, for use in bridging applications etc. (See figure below)



Installation

The AnyBus-S API is installed automatically when installing a supported interface. However, in order to be able to use the API, some library files must be included in the compiler / interpreter. Consult the compiler documentation for more information on how to include these files in your project.

- **Visual C++**

- ABS.H Header file
- ABS.LIB Library file

- **Borland C++**

- ABS.H Header file
- ABS.LIB Library file

- **Visual Basic**

It is possible to access the API from Visual Basic, however no library files are supplied by HMS at the time of writing.

- **Delphi**

It is possible to access the API from Delphi, however no library files are supplied by HMS at the time of writing.

AnyBus-S Overview

This section gives a brief introduction to the AnyBus-S platform. However, in order to be able to exploit the full potential of the AnyBus-S communication module, please consult the general AnyBus-S Design Guide.

Basic Concept

The AnyBus-S is a series of interchangeable fieldbus communication modules featuring on board memory and processing power. All software and hardware functionality required to communicate on the fieldbus is included in the module itself, allowing the application to focus on other tasks.

The AnyBus-S communication module is based on a dual port memory architecture¹, where the application (in this case the software application and interface drivers) and the AnyBus-S can access the same data simultaneously via a shared memory area, see below.

Dual Port Memory Contents (Shared Memory Area)

The Dual Port Memory is sub divided in several areas based on their function, see below. For more information about these areas, consult the general AnyBus-S Design Guide.

- **Input / Output Data Area**
This area contains I/O data from / to the fieldbus. The API features dedicated function calls for reading and writing to this area.
- **Mailbox Input / Output Area**
The mailbox area is used for acyclic communication and configuration of the AnyBus-S module. The API features dedicated function calls for sending and receiving mailbox commands.
- **Fieldbus Area**
This area contains fieldbus specific information, and the contents vary between different AnyBus-S modules. The API features dedicated function calls for reading and writing to this area.
- **Control Register Area**
This area contains various fieldbus control and status bits. The API features dedicated function calls for reading and writing to this area.
- **Handshake Register Area**
This area features various handshaking bits used to ensure data constancy between the application and the AnyBus-S. When using the AnyBus-S API functions, all handshaking is performed automatically using locked request / unlocked release. (For more information about the handshaking process, consult the general AnyBus-S Design Guide)

1. Some AnyBus-S modules are available with an asynchronous serial interface instead of a dual port memory. These modules are not supported by the AnyBus-S API at this point, and is not the scope of this document.

Initialisation Process

Before any fieldbus communication can take place, the AnyBus-S module must be initialised. The initialisation process is mandatory and decides how the module should operate on the network. Initialisation is performed using mailbox commands, and is described thoroughly in the general AnyBus-S Design Guide.

Basically, the initialisation sequence looks like this:

■ Power On (Reset)

1. Send mailbox command 'Start Init'

This step starts the initialisation sequence.

2. Send mailbox command 'FB Init'

This step is optional, and is used to initialise some of the more advanced AnyBus-S functions.

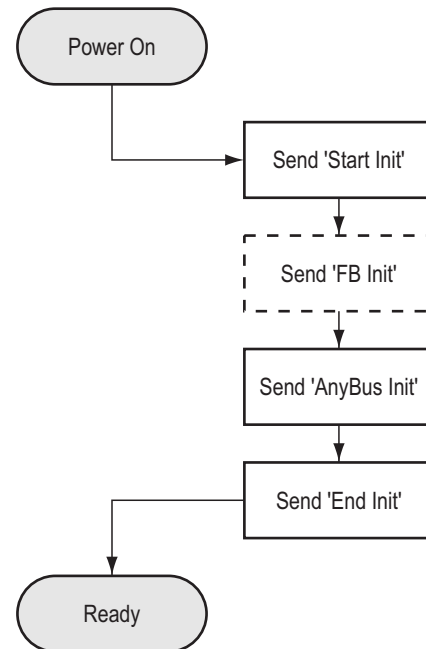
3. Send mailbox command 'AnyBus Init'

'AnyBus Init' is used to set up I/O sizes and various configuration bits.

4. Send mailbox command 'End Init'

This step ends the initialisation sequence.

■ Ready



Basic Operation

General

Byte Swapping

The AnyBus communication modules uses Motorola format (big-endian) for all 16/32 bit values. However in order to save the programmer from “re-inventing the wheel”, the API will perform the necessary byte-swapping automatically.

The functions performing byte-wise storage in the AnyBus memory space cannot determine whether the intention is byte or word access and will therefore never perform any such byte swapping. Keep this in mind when using such functions, e.g. `ABS_ReadControlArea()`.

Timeout Values

Many of the API functions requires a timeout parameter to avoid locking up the application in the case of an error. Although the implementation describes the timeout time in ms, the resolution might differ due to the nature of the different Windows versions. Even though the API tries to set a timer resolution of one ms for the application, this timer resolution cannot be guaranteed.

The default timeout times for the different operating systems are listed below:

| Operating System: | Default Timeout |
|-------------------|-----------------|
| Windows 98 | 54.95 ms |
| Windows ME | 54.95 ms |
| Windows NT | 10 ms |
| Windows 2000 | 10 ms |
| Windows XP | 10 ms |

When determining timeout times for your application, please also consider the fact that the different transport paths may need different amounts of time to perform the desired operations, e.g. there is a big performance difference between reading a mailbox message via the PCI bus compared to reading it via a serial link at 4800bits/s.

Notes About Multitasking Programming Models

Programming in a multi tasking environment requires understanding of many issues which do not exist in a single threaded environment. These issues can be especially complex when involving hardware device drivers. Therefore, this API has been designed for a single tasking environment. In other words, the library should be considered as non re-entrant and there is no use of locking mechanism for hardware access and internal global variables.

The issue of multi tasking is especially complex with regards to the AnyBus-S module, since certain operations requires the AnyBus module to remain in a specific state until the operation has completed. (E.g handshaking, read/write and release of areas is on example of this).

Path Management

Before the application can gain access to an interface, a transport path must be opened towards it. All actions performed towards the interface are then passed through the transport path, i.e. it forms a link between the application and the physical interface. If the system contains more than one supported physical interface, multiple paths can be opened. Depending on interface type, it may or may not be possible to open more than one path per interface.

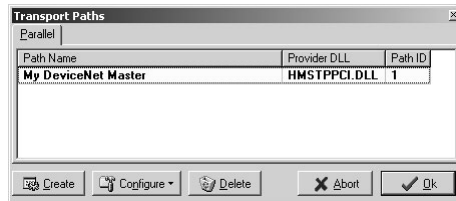
Once a transport path has been opened, the application can access the AnyBus communication module using the functions described later in this document. (See 4 “Functions”)

Path Related Functions

The following functions described handles transport paths. For more information, see 4-2 “Path Management”.

- **ABS_UserSelectPath()**

This function displays a modal form prompting the user for a transport path to use. The form also permits the user to configure the transport path¹.



The function returns a Path ID that can be used to identify the path later, and a pointer to a path identifier structure, that should be used on all further function calls associated with that path.

- **ABS_SelectPath()**

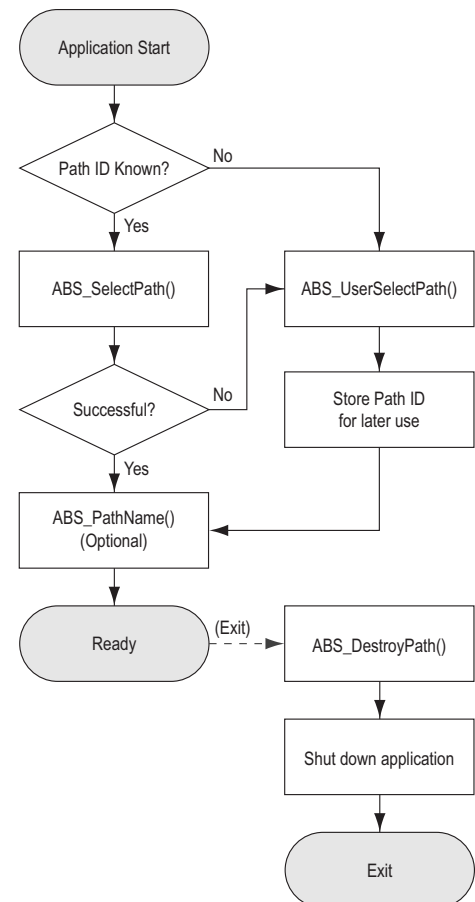
This function opens a path with a known PathID. The function returns a pointer to a path identifier structure, that should be used on all further function calls associated with that path.

- **ABS_PathName()**

This function returns the name of a previously opened path.

- **ABS_DestroyPath()**

When the path should not be used any more, i.e. when exiting the application, the path must be closed using ABS_DestroyPath().



1. In Windows XP/2000/NT this requires that the user has administration rights.

Example

The following example will prompt the user for a transport path, and store the path ID and path identifier pointer in lPathId and pxMyPath respectively.

```
ABS_StatusType    eStatus;
UINT32            lPathId;
void*             pxMyPath;

/*
** Ask the user which path should be used.
*/

eStatus = ABS_UserSelectPath( &lPathId, &pxMyPath );

if( eStatus == TP_ERR_NONE )
{
    /*
    ** Path Opened successfully
    */
}
else if( eStatus == TP_ERR_ABORTED )
{
    /*
    ** The user pressed the abort button
    */
}
else
{
    /*
    ** An error occurred
    */
}
```

Functions

The functions described in this chapter are grouped by their usage according to below:

- **Path Management**

These functions are used to create, open, configure, and manage paths.

| Function | Description | Page |
|----------------------|---|------|
| ABS_UserSelectPath() | Prompts the user for a transport path, and opens the path | 4-2 |
| ABS_SelectPath() | Opens a transport path with a known Path ID | 4-4 |
| ABS_PathName() | Returns a pointer to the name of the specified path | 4-5 |
| ABS_DestroyPath() | Closes a previously opened path | 4-6 |

- **Mailbox Functions**

These functions are used to handle mailbox communication towards the AnyBus module.

| Function | Description | Page |
|-------------------|--|------|
| ABS_SendMail() | Sends a mailbox message to the AnyBus communication module | 4-7 |
| ABS_ReceiveMail() | Reads a mailbox message from the AnyBus communication module | 4-8 |

- **Read/Write Operations**

These functions are used to read and write data to the AnyBus module.

| Function | Description | Page |
|-------------------------|---|------|
| ABS_ReadControlArea() | Reads data from the AnyBus control area | 4-9 |
| ABS_WriteControlArea() | Writes data to the AnyBus control area | 4-10 |
| ABS_ReadFieldbusArea() | Reads data from the fieldbus specific area of the AnyBus module | 4-11 |
| ABS_WriteFieldbusArea() | Writes data to the fieldbus specific area of the Anybus module | 4-12 |
| ABS_ReadInArea() | Reads data from the AnyBus input data area | 4-13 |
| ABS_WriteInArea() | Writes data to the AnyBus input data area | 4-14 |
| ABS_ReadOutArea() | Reads data from the AnyBus output data area | 4-15 |

- **Miscellaneous**

| Function | Description | Page |
|-------------------------|---|------|
| ABS_ModuleInfo() | Returns general information about the AnyBus module | 4-16 |
| ABS_ModuleStatus() | Returns status information from the AnyBus communication module | 4-17 |
| ABS_SetDpramSize() | Enables support for 4kb dpram, used by some AnyBus modules | 4-18 |
| ABS_Reset() | Attempts to reset the AnyBus module. | 4-19 |
| ABS_SetApplicationLed() | Controls the Application LED on some AnyBus modules. | 4-20 |

Note: The functions described in this chapter performs all necessary handshaking automatically using locked request/unlocked release. However, in some applications it may be preferable to access the AnyBus memory space directly and perform all handshaking manually, see Appendix A-1 “Advanced Functions”.

Path Management

ABS_UserSelectPath()

Syntax:

```
ABS_StatusType ABS_UserSelectPath( UINT32*   pIPathId,
                                   void**    pPxPath );
```

Description:

This function displays a modal form prompting the user for the transport path to use. The user will also be able to create, configure and delete paths using this form.

Notes:

Both this function and *ABS_SelectPath()* opens and configures the path, i.e. only one of the functions should be called to open a path. The PathId returned by this function may be used together with *ABS_SelectPath()* to use the same path the next time.

When the application is about to terminate, or does not need to perform more actions towards the path, it shall be closed again with the function *ABS_DestroyPath()*.

Parameters:

| Parameter | Description |
|-----------|---|
| pIPathId | Pointer to variable which, if the function succeeds, will contain the PathId the user selected. |
| ppxPath | Pointer to path identifier pointer, which will be set by the function if the function succeeds. The pxPath shall then be used in latter calls into the API to identify the path upon which to perform the action. |

Usage:

```

ABS_StatusType    eStatus;
UINT32            lPathId;
void*             pxMyPath;

/*
** Ask the user which path should be used.
*/

eStatus = ABS_UserSelectPath( &lPathId, &pxMyPath );

if( eStatus == TP_ERR_NONE )
{
    /*
    ** Path Opened successfully
    */
}
else if( eStatus == TP_ERR_ABORTED )
{
    /*
    ** The user pressed the abort button
    */
}
else
{
    /*
    ** An error occurred
    */
}

```

Related Information:

| Information | Page |
|-------------------|------|
| ABS_DestroyPath() | 4-6 |
| ABS_SelectPath() | 4-4 |
| ABS_StatusType | 5-5 |

ABS_SelectPath()

Syntax:

```
ABS_StatusType ABS_SelectPath( UINT32    lPathId,
                               void**   ppxPath );
```

Description:

This function could be called by the application when the *lPathId* is known. The path will then be opened and configured (assuming it exists).

Notes:

Both this function and *ABS_UserSelectPath()* opens and configures the path, i.e. only one of the functions should be called to open a path.

When the application is about to terminate, or does not need to perform more actions towards the path, it shall be closed again with the function *ABS_DestroyPath()*.

Parameters:

| Parameter | Description |
|-----------|--|
| lPathId | Id of the path to be opened. |
| ppxPath | Pointer to path identifier pointer, which will be set by the function if the function succeeds. The ppxPath shall then be used in latter calls into the API to identify the path upon which to perform the action. |

Usage:

```
ABS_StatusType    eStatus;
void*             ppxMyPath;

/*
** Open path with id 31
*/

eStatus = ABS_SelectPath( 31, &ppxMyPath );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to open the path
    */
}
```

Related Information:

| Information | Page |
|----------------------|------|
| ABS_DestroyPath() | 4-6 |
| ABS_UserSelectPath() | 4-2 |
| ABS_StatusType | 5-5 |

ABS_PathName()

Syntax:

```
ABS_StatusType ABS_PathName( void*    pxPath,
                             char**   ppbPathName );
```

Description:

This function returns a pointer to the user supplied name of the path. The pointer is valid until ABS_DestroyPath has been called.

Parameters:

| Parameter | Description |
|-------------|--|
| pxPath | Path identifier pointer of the path from which the name shall be extracted. |
| ppbPathName | Pointer to char pointer which will be set to the name of the path as a null-terminated string. |

Usage:

```
ABS_StatusType    eStatus;
char*             pbPathName;

/*
** Set pointer to the paths name
*/

eStatus = ABS_PathName( pxPath, &pbPathName );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to retrieve path name
    */
}

/*
** Show the name to the user
*/

printf( "Path Name: %s", pbPathName );
```

Related Information:

| Information | Page |
|----------------|------|
| ABS_StatusType | 5-5 |

ABS_DestroyPath()

Syntax:

```
ABS_StatusType ABS_DestroyPath( void* pxPath );
```

Description:

This function frees previously allocated memory used to identify a transport path, i.e. a path opened using either *ABS_SelectPath()* or *ABS_UserSelectPath()*.

Notes:

After calling this function the 'pxPath' pointer will be invalid for use with further calls to the API.

Parameters:

| Parameter | Description |
|-----------|---|
| pxPath | Path identifier pointer describing the path to be destroyed |

Usage:

```
ABS_StatusType eStatus;

eStatus = ABS_DestroyPath( pxMyPath );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Error - Unable to close the path
    */
}
```

Related Information:

| Information | Page |
|----------------------|------|
| ABS_StatusType | 5-5 |
| ABS_SelectPath() | 4-4 |
| ABS_UserSelectPath() | 4-2 |

Mailbox Functions

ABS_SendMail()

Syntax:

```
ABS_StatusType ABS_SendMail( void*          pxPath,
                             ABS_MailboxType* psMail,
                             UINT16         iTimeout );
```

Description:

This function attempts to send a mailbox message to an AnyBus module. The function waits for no more than the *iTimeout* specified for the AnyBus to indicate that it is ready to receive another mailbox message, and if possible, sends the specified message (*psMail*) to the AnyBus module.

Notes:

Most of the mailbox messages that are sent triggers a response message to be sent from the AnyBus. Therefore, upon sending a mailbox message, be sure to await possible response messages issued by the AnyBus module.

Parameters:

| Parameter | Description |
|-----------|--|
| pxPath | Path identifier pointer to the AnyBus the mail is intended for. |
| psMail | Pointer to structure of the ABS_MailboxType containing the mail to be sent. |
| iTimeout | The maximum time the application is willing to wait for the mail to be sent. |

Usage:

```
ABS_StatusType    eStatus;
ABS_MailboxType   sMail;

/*
** Build and send a mail
*/

sMail.iId = 0x0001;

/* Keep filling out the mail */

eStatus = ABS_SendMail( pxPath, &sMail, 50 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to send the mail
    */
}
```

Related Information:

| Information | Page |
|-------------------|------|
| ABS_ReceiveMail() | 4-8 |
| ABS_MailboxType | 5-1 |
| ABS_StatusType | 5-5 |

ABS_ReceiveMail()

Syntax:

```
ABS_StatusType ABS_ReceiveMail( void*          pxPath,
                                ABS_MailboxType* psMail,
                                UINT16         iTimeout );
```

Description:

This function attempts to receive a mailbox message from an AnyBus module. The function waits no more than the *iTimeout* specified for a mailbox message from the AnyBus module and, if one is available, stores it into the *ABS_MailboxType* structure indicated by *psMail* and thereafter indicates to the AnyBus module that the message has been read.

Notes:

This function will check for a new mailbox message once regardless of what the timeout time is set to. Therefore, in order to quickly check for a new mail, set the timeout time to 0.

Parameters:

| Parameters | Description |
|------------|--|
| pxPath | Path identifier pointer to the AnyBus module from whom the mail is expected. |
| psMail | Pointer to structure of the <i>ABS_MailboxType</i> where a possibly received mail shall be stored. |
| iTimeout | The maximum time the application is willing to wait for a mail. |

Usage:

```
ABS_StatusType    eStatus;
ABS_MailboxType  sMail;

/*
** Check for new mail
*/

eStatus = ABS_ReceiveMail( pxPath, &sMail, 0 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Function failed
    ** Possibly it timed out while waiting for mail
    */
}
```

Related Information:

| Information | Page |
|-----------------|------|
| ABS_SendMail() | 4-7 |
| ABS_MailboxType | 5-1 |
| ABS_StatusType | 5-5 |

Read/Write Operations

ABS_ReadControlArea()

Syntax:

```
ABS_StatusType ABS_ReadControlArea( void*    pxPath,
                                     UINT16   iOffset,
                                     UINT8*    pbData,
                                     UINT16   iAmount,
                                     UINT16   iTimeout );
```

Description:

This function waits no more than the *iTimeout* specified to try and get access to the control area of the AnyBus module. If access is granted, it stores the amount of desired data from *iOffset* in the control area to the position indicated by the *pbData* pointer. It then releases the area.

Notes:

Consult the general AnyBus-S Design Guide for more information regarding the Control Area.

Parameters:

| Parameters | Description |
|------------|--|
| pxPath | Path identifier pointer to the AnyBus module whose control area should be read. |
| iOffset | Offset in the control area to read from. |
| pbData | Pointer indicating where to store the read bytes. |
| iAmount | The amount of bytes to read. |
| iTimeout | The maximum time the application is willing to wait for the data exchange to complete. |

Usage:

```
ABS_StatusType  eStatus;
UINT8          bTopLeftLed;

/*
** Read out the status of the top left led
** It is located at offset 0x1A in the control area
*/

eStatus = ABS_ReadControlArea( pxPath, 0x1A, &bTopLeftLed, 1, 100 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to read the led status
    */
}
```

Related Information:

| Information | Page |
|----------------|------|
| ABS_StatusType | 5-5 |

ABS_WriteContolArea()

Syntax:

```
ABS_StatusType ABS_WriteContolArea( void*    pxPath,
                                     UINT16   iOffset,
                                     UINT8*    pbData,
                                     UINT16   iAmount,
                                     UINT16   iTimeout );
```

Description:

This function waits no more than the *iTimeout* specified to try and get access to the control area of the AnyBus module. If access is granted, it writes the amount of desired data to *iOffset* in the control area from the position indicated by the *pbData* pointer. It then releases the area.

Notes:

Consult the general AnyBus-S Design Guide for more information regarding the Control Area.

Parameters:

| Parameter | Description |
|-----------|--|
| pxPath | Path identifier pointer to the AnyBus module whose control area should be written. |
| iOffset | Offset in the in area to write to. |
| pbData | Pointer indicating where to fetch bytes to be written. |
| iAmount | The amount of bytes to written. |
| iTimeout | The maximum time the application is willing to wait for the data exchange to complete. |

Usage:

```
ABS_StatusType    eStatus;
UINT16            iWatchdog;

/*
** Write to the Watchdog counter in (offset 0x12)
** assuming in this example that iWatchdog contains valid value
*/

eStatus = ABS_WriteControlArea( pxPath, 18, &iWatchdog, 2, 100 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to write the data
    */
}
```

Related Information:

| Information | Page |
|----------------|------|
| ABS_StatusType | 5-5 |

ABS_ReadFieldbusArea()

Syntax:

```
ABS_StatusType ABS_ReadFieldbusArea( void*    pxPath,
                                     UINT16   iOffset,
                                     UINT8*    pbData,
                                     UINT16   iAmount,
                                     UINT16   iTimeout );
```

Description:

This function waits no more than the *iTimeout* specified to try and get access to the fieldbus area of the AnyBus module. If access is granted, it stores the amount of desired data from *iOffset* in the fieldbus area to the position indicated by the *pbData* pointer. It then releases the area.

Notes:

For information regarding the data contained in the fieldbus specific area, please consult the appropriate fieldbus appendix.

Parameters:

| Parameter | Description |
|-----------|--|
| pxPath | Path identifier pointer to the AnyBus module whose control area should be read. |
| iOffset | Offset in the fieldbus area to read from. |
| pbData | Pointer indicating where to store the read bytes. |
| iAmount | The amount of bytes to read. |
| iTimeout | The maximum time the application is willing to wait for the data exchange to complete. |

Usage:

```
ABS_StatusType  eStatus;
UINT8          abData[ 2 ];

/*
** Read out the first two bytes in the fieldbus specific area
*/

eStatus = ABS_ReadFieldbusArea( pxPath, 0x00, abData, 2, 100 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to read the data
    */
}
```

Related Information:

| Information | Page |
|----------------|------|
| ABS_StatusType | 5-5 |

ABS_WriteFieldbusArea()

Syntax:

```
ABS_StatusType ABS_WriteFieldbusArea( void*    pxPath,
                                       UINT16   iOffset,
                                       UINT8*    pbData,
                                       UINT16   iAmount,
                                       UINT16   iTimeout );
```

Description:

This function waits no more than the *iTimeout* specified to try and get access to the fieldbus area of the AnyBus module. If access is granted, it writes the amount of desired data to *iOffset* in the fieldbus area from the position indicated by the *pbData* pointer. It then releases the area.

Notes:

For information regarding the data contained in the fieldbus specific area, please consult the appropriate fieldbus appendix.

Parameters:

| Parameter | Description |
|-----------|--|
| pxPath | Path identifier pointer to the AnyBus module whose fieldbus specific area should be written. |
| iOffset | Offset in the in area to write to. |
| pbData | Pointer indicating where to fetch bytes to be written. |
| iAmount | The amount of bytes to written. |
| iTimeout | The maximum time the application is willing to wait for the data exchange to complete. |

Usage:

```
ABS_StatusType    eStatus;
UINT16            abData[ 2 ];

/*
** Write to the first two bytes in the fieldbus specific area
** assuming in this example that abData contains values to write
*/

eStatus = ABS_WriteFieldbusArea( pxPath, 0, abData, 2, 100 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to write the data
    */
}
```

Related Information:

| Information | Page |
|----------------|------|
| ABS_StatusType | 5-5 |

ABS_ReadInArea()

Syntax:

```

ABS_StatusType ABS_ReadInArea( void*    pxPath,
                               UINT16   iOffset,
                               UINT8*    pbData,
                               UINT16   iAmount,
                               UINT16   iTimeout );
    
```

Description:

This function waits no more than the *iTimeout* specified to try and get access to the in area of the AnyBus module. If access is granted, it stores the amount of desired data from *iOffset* in the in area to the position indicated by the *pbData* pointer. It then releases the area.

Notes:

The in area is the data area in which the application stores data that should be transmitted onto the fieldbus. With this function it is possible to read back previously written data.

Parameters:

| Parameter | Description |
|-----------|--|
| pxPath | Path identifier pointer to the AnyBus module whose in area should be read. |
| iOffset | Offset in the in area to read from. |
| pbData | Pointer indicating where to store the read bytes. |
| iAmount | The amount of bytes to read. |
| iTimeout | The maximum time the application is willing to wait for the data exchange to complete. |

Usage:

```

ABS_StatusType  eStatus;
UINT8           abData[ 2 ];

/*
** Read out the first two bytes in the in area
*/

eStatus = ABS_ReadInArea( pxPath, 0, abData, 2, 100 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to read the data
    */
}
    
```

Related Information:

| Information | Page |
|----------------|------|
| ABS_StatusType | 5-5 |

ABS_WriteInArea()

Syntax:

```
ABS_StatusType ABS_WriteInArea( void*    pxPath,
                                UINT16   iOffset,
                                UINT8*    pbData,
                                UINT16   iAmount,
                                UINT16   iTimeout );
```

Description:

This function waits no more than the *iTimeout* specified to try and get access to the in area of the AnyBus module. If access is granted, it writes the amount of desired data to *iOffset* in the in area from the position indicated by the *pbData* pointer. It then releases the area.

Notes:

The in area is the data area in which the application stores data that should be transmitted onto the fieldbus. Thus with this function it is possible to produce data onto the network.

Parameters:

| Parameter | Description |
|-----------|--|
| pxPath | Path identifier pointer to the AnyBus module whose in area should be written. |
| iOffset | Offset in the in area to write to. |
| pbData | Pointer indicating where to fetch bytes to be written. |
| iAmount | The amount of bytes to written. |
| iTimeout | The maximum time the application is willing to wait for the data exchange to complete. |

Usage:

```
ABS_StatusType  eStatus;
UINT16          abData[ 16 ];

/*
** Write to the start of the in area
** assuming in this example that abData contains values to write
*/

eStatus = ABS_WriteInArea( pxPath, 0, abData, 16, 100 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to write the data
    */
}
```

Related Information:

| Information | Page |
|----------------|------|
| ABS_StatusType | 5-5 |

ABS_ReadOutArea()

Syntax:

```
ABS_StatusType ABS_ReadOutArea( void*   pxPath,
                                UINT16  iOffset,
                                UINT8*   pbData,
                                UINT16  iAmount,
                                UINT16  iTimeout );
```

Description:

This function waits no more than the *iTimeout* specified to try and get access to the out area of the AnyBus module. If access is granted, it stores the amount of desired data from *iOffset* in the out area to the position indicated by the *pbData* pointer. It then releases the area.

Notes:

The out area is the data area in which the AnyBus stores data that is received from the fieldbus.

Parameters:

| Parameter | Description |
|-----------|--|
| pxPath | Path identifier pointer to the AnyBus module whose out area should be read. |
| iOffset | Offset in the in area to read from. |
| pbData | Pointer indicating where to store the read bytes. |
| iAmount | The amount of bytes to read. |
| iTimeout | The maximum time the application is willing to wait for the data exchange to complete. |

Usage:

```
ABS_StatusType  eStatus;
UINT8          abData[ 2 ];

/*
** Read out the first two bytes in the out area
*/

eStatus = ABS_ReadOutArea( pxPath, 0, abData, 2, 100 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to read the data
    */
}
```

Related Information:

| Information | Page |
|----------------|------|
| ABS_StatusType | 5-5 |

Miscellaneous

ABS_ModuleInfo()

Syntax:

```
ABS_StatusType ABS_ModuleInfo( void*          pxPath,
                               ABS_ModuleInfoType* psInfo,
                               UINT16          iTimeout );
```

Description:

This function attempts to read out a structure containing information about the AnyBus module. The structure consists of items that do not change during runtime, such as version numbers etc.

Notes:

Make sure that the iSize element is set to the size of the structure prior to calling this function.

Parameters:

| Parameter | Description |
|-----------|--|
| pxPath | Path identifier pointer to the path from which the psInfo structure should be filled out |
| psInfo | Pointer to ABS_ModuleInfoType structure where the first element (iSize) is initialised to the size of the structure (sizeof(ABS_ModuleInfoType)). If successful the function will update the information in the structure. |
| iTimeout | The maximum time the application is willing to wait for the structure to be updated. |

Usage:

```
ABS_StatusType      eStatus;
ABS_ModuleInfoType  sInfo;

sInfo.iSize = sizeof( ABS_ModuleInfoType );
eStatus = ABS_ModuleInfo( pxPath, &sInfo, 20 );

if( eStatus != TP_ERR_NONE )
{
    /*
     ** Failed to update the sInfo structure
     */
}
```

Related Information:

| Information | Page |
|--------------------|------|
| ABS_ModuleInfoType | 5-3 |
| ABS_StatusType | 5-5 |

ABS_ModuleStatus()

Syntax:

```
ABS_StatusType ABS_ModuleStatus( void*          pxPath,
                                ABS_ModuleStatusType* psStatus,
                                UINT16          iTimeout );
```

Description:

This function attempts to read out a structure containing information about the AnyBus module. The structure consists of items that are subject to changes during runtime, such as watchdog etc.

Notes:

Make sure that the iSize element is set to the size of the structure prior to calling this function.

Parameters:

| Parameter | Description |
|-----------|--|
| pxPath | Path identifier pointer to the path from which the psStatus structure should be filled out. |
| psStatus | Pointer to ABS_ModuleStatusType structure where the first element (iSize) is initialised to the size of the structure (sizeof(ABS_ModuleStatusType)). If successful the function will update the information in the structure. |
| iTimeout | The maximum time the application is willing to wait for the structure to be updated. |

Usage:

```
ABS_StatusType          eStatus;
ABS_ModuleStatusType    sModuleStatus;

sModuleStatus.iSize = sizeof( ABS_ModuleStatusType );
eStatus = ABS_ModuleStatus( pxPath, &ModuleStatus, 20 );

if( eStatus != TP_ERR_NONE )
{
    /*
     ** Failed to update the sModuleStatus structure
     */
}
```

Related Information:

| Information | Page |
|----------------------|------|
| ABS_ModuleStatusType | 5-4 |
| ABS_StatusType | 5-5 |

ABS_SetDpramSize()

Syntax:

```
ABS_StatusType ABS_SetDpramSize( void*   pxPath,
                                UINT16  iSize );
```

Description:

This function is used to enable support for AnyBus modules with more than 2kb dual port memory such as the AnyBus Profibus DPV master. Note that the AnyBus module has to be initialised to extended dpram mode prior to calling this function.

Notes:

By default, the API uses a 2kb dual port memory map which is compatible with the default initialisations of all AnyBus modules.

Parameters:

| Parameter | Description |
|-----------|---|
| pxPath | Path identifier pointer on which to change the dual port memory structure |
| iSize | The size of the dual port memory in bytes |

Usage:

```
ABS_StatusType   eStatus;

/*
** Initialisation complete. AnyBus should now be in
** 4kb dual port memory mode. Alter API accordingly
*/

eStatus = ABS_SetDpramSize( pxPath, 4096 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to alter size of dual port memory
    */
}
```

Related Information:

| Information | Page |
|----------------|------|
| ABS_StatusType | 5-5 |

ABS_Reset()

Syntax:

```
ABS_StatusType ABS_Reset( void*   pxPath,
                          UINT8   bType,
                          UINT16  iTimeout );
```

Description:

This function controls the reset line towards the AnyBus module, providing that the transport path is able to support one. If the transport path is unable to control the reset line, the function will return the ABS_ERR_UNSUPPORTED status code.

Notes:

Certain AnyBus modules have long start-up times and it is therefore recommended to have a long *iTimeout* value, e.g. 10000 (10s).

Parameters:

| Parameter | Description |
|-----------|--|
| pxPath | Path identifier pointer to the AnyBus module whose reset line shall be controlled. |
| bType | Type of desired action. See also 6-10 "ABS_RESET_xxx" |
| iTimeout | If a bType of ABS_RESET_MODULE is selected, then the function shall wait for the AnyBus module to become ready to communicate before returning. It will not wait longer than the iTimeout time though. |

Usage:

```
ABS_StatusType  eStatus;

/*
** Perform a hardware reset of the module
*/

eStatus = ABS_Reset( pxPath, ABS_RESET_MODULE, 10000 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to reset the module
    */
}
```

Related Information:

| Information | Page |
|----------------|------|
| ABS_StatusType | 5-5 |
| ABS_RESET_xxx | 6-10 |

ABS_SetApplicationLed()

Syntax:

```
ABS_StatusType ABS_SetApplicationLed( void*    pxPath,
                                     UINT8    bLed,
                                     UINT8    bColour );
```

Description:

This function tries to set the desired colour to the specified application led, providing that it exists on the selected path. If the transport path is unable to control the led, the function will return the `ABS_ERR_UNSUPPORTED` status code.

Parameters:

| Parameter | Description |
|-----------|---|
| pxPath | Path identifier pointer to the hardware with the led to be controlled. |
| bLed | The led to try and control. First available led should be indexed as 0. |
| bColour | The desired color. See also 6-4 "ABS_LED_xxx" |

Usage:

```
ABS_StatusType  eStatus;

/*
** Set the first application led green
*/

eStatus = ABS_SetApplicationLed( pxPath, 0, ABS_LED_GREEN );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to set the led
    */
}
```

Related Information:

| Information | Page |
|----------------|------|
| ABS_StatusType | 5-5 |
| ABS_LED_xxx | 6-4 |

Data Structures and Enumerations

ABS_MailboxType

Definition:

```
typedef struct ABS_MailboxType
{
    UINT16    iId;
    UINT16    iInfo;
    UINT16    iCommand;
    UINT16    iDataSize;
    UINT16    iFrameCount;
    UINT16    iFrameNo;
    UINT16    iOffsetHi;
    UINT16    iOffsetLo;

    UINT16    aiExtended[ 8 ];
    UINT8     abData[ 256 ];
}
```

Description:

This structure is intended to be used with the *ABS_ReceiveMail()* and the *ABS_SendMail()* functions and describe the mailbox messages according to the AnyBus-S Design Guide. For detailed information about the AnyBus-S mailbox interface, please consult the AnyBus-S Design Guide.

Members:

| Member | Description |
|-------------|---|
| iId | Contains an integer identifier for the command. If several mailboxes are sent to the AnyBus using the same message type and command, this member can be used to identify which of the requests the received mail is a response to. When responding to mails issued by the AnyBus, use the same identifier in the response as the AnyBus module used in the request. |
| iInfo | This member contains bit and code information about the mailbox message. See also 6-5 "ABS_MB_xxx" and the AnyBus-S Design Guide for further information. |
| iCommand | This member contains the command identifier. |
| iDataSize | The data size register indicates the number of valid bytes in the message data area. Valid data sizes are 0 – 256 bytes. |
| iFrameCount | The frame count register indicates the total number of frames in the message. If the number of frames is unknown, the member contents must be zero. A normal mailbox message consists of only one frame. |
| iFrameNo | The frame number indicates the current frame number. This member must be incremented with one for each frame sent within the fragmented message. The register must be incremented even if the number of frames is unknown. In the case of a normal one-frame message, this member is set to one. |
| iOffsetHi | This member contains the offset address for the data associated with the current frame. |
| iOffsetLo | This information can be used to double-check the mailbox handling and to unload the application of certain calculations. For a normal one-frame message, this member is set to zero. |
| aiExtended | The extended header is specific for each command. The data herein is only used by some of the available commands. Please see the specification for each command. |
| abData | This is the data portion of the mailbox message. It contains data associated with the command. Please see the specification of the command for a description of the data for the different mailboxes. |

Related Information:

| Information | Page |
|-------------------|------|
| ABS_ReceiveMail() | 4-8 |
| ABS_SendMail() | 4-7 |
| ABS_MB_xxx | 6-5 |

ABS_ModuleInfoType

Definition:

```
typedef struct ABS_ModuleInfoType
{
    UINT16    iSize;
    UINT16    iModuleType;
    UINT16    iFieldbusType;
    UINT16    iVendorId;
    UINT32    lSerialNumber;
    UINT16    iBootloaderVersion;
    UINT16    iInterfaceVersion;
    UINT16    iFieldbusVersion;
    UINT16    iModuleVersion;
    UINT8     bModuleBuild;
}
ABS_ModuleInfoType;
```

Description:

This structure is used in conjunction with the `ABS_ModuleInfo()` function in order to read out registers considered as static from the AnyBus module, i.e. registers which are not updated during runtime.

Members:

| Member | Description |
|--------------------|--|
| iSize | This element shall be set to the size of the structure prior to the call of <i>ABS_ModuleInfo()</i> . |
| iModuleType | Identifies the type of AnyBus module. See also 6-8 "ABS_MODULE_TYPE_XXX" |
| iFieldbusType | Identifies the fieldbus type of the AnyBus module. See also 6-2 "ABS_FB_XXX" |
| iVendorId | Identifies the vendor of the module: 0x0001 – HMS Industrial Networks AB |
| lSerialNumber | Unique 32-bit serial number of the module |
| iBootloaderVersion | Identifies the bootloader version of the module. E.g. 0x122 equals version 1.22. Note: If bootloader version is not implemented in the module this element may equal 0x0000. |
| iInterfaceVersion | Identifies the version of the code handling the application interface. E.g. 0x0100 equals version 1.00. Note: If interface version is not implemented in the module this element may equal 0x0000. |
| iFieldbusVersion | Identifies the version of the code handling the fieldbus protocol. E.g. 0x0101 equals version 1.01. Note: If fieldbus version is not implemented in the module this element may equal 0x0000. |
| iModuleVersion | Identifies the total overall version of the module. E.g. 0x0120 equals version 1.20. |
| bModuleBuild | Identifies the build number of the overall version and is used in conjunction with the iModuleVersion field. |

Related Information:

| Information | Page |
|---------------------|------|
| ABS_ModuleInfo() | 4-16 |
| ABS_FB_XXX | 6-2 |
| ABS_MODULE_TYPE_XXX | 6-8 |

ABS_ModuleStatusType

Definition:

```
typedef struct ABS_ModuleStatusType
{
    UINT16    iSize;
    UINT16    iWatchdog;
    UINT8     abLedStatus[ 6 ];
    UINT16    iModuleStatus;
    UINT8     fInitialised;
}
ABS_ModuleStatusType;
```

Description:

This structure is used in conjunction with the *ABS_ModuleStatus()* function in order to read out registers considered as dynamic from the AnyBus module, i.e. registers which will be subsequent to changes during runtime.

Members:

| Member | Description |
|---------------|--|
| iSize | This element shall be set to the size of the structure prior to the call of <i>ABS_ModuleStatus()</i> . |
| iWatchdog | Contains a counter which counts in ms and is incremented by the AnyBus module. Please note that the maximum update time of this counter is 50 ms, i.e. the value may be incremented with a maximum of 50. |
| abLedStatus | Contains the statuses of the Led's. Note: The descriptions of the Led positions below refer to their description in the AnyBus appendices for the respective fieldbus. Also keep in mind that the PCI card is most often mounted upside down in a PC with regards to the Led's. abLedStatus[0] - Top Left Led according to the appendix abLedStatus[1] - Top Right Led according to the appendix abLedStatus[2] - Bottom Left Led according to the appendix abLedStatus[3] - Bottom Right Led according to the appendix abLedStatus[4 & 5]- Reserved for future use For colour information please see the 6-4 "ABS_LED_xxx". |
| iModuleStatus | Please refer to the AnyBus-S Design Guide for further information regarding this register. See also 6-9 "ABS_MODULE_STATUS_xxx". |
| finitialised | This flag is zero if the module is not yet initialised. Any other value means the module has been initialised. |

Related Information:

| Information | Page |
|-----------------------|------|
| ABS_ModuleStatus() | 4-17 |
| ABS_LED_xxx | 6-4 |
| ABS_MODULE_STATUS_xxx | 6-9 |

ABS_StatusType

Definition:

```
typedef enum ABS_StatusType
{
    TP_ERR_NONE                = 0x0000,
    TP_ERR_OTHER               = 0x0001,
    TP_ERR_WIN_FAIL           = 0x0002,
    TP_ERR_MEM_ALLOC          = 0x0003,
    TP_ERR_CONFIG             = 0x0004,
    TP_ERR_ABORTED           = 0x0005,
    TP_ERR_MEM_SIZE           = 0x0006,
    TP_ERR_OPEN               = 0x0007,
    TP_ERR_NO_HW              = 0x0008,
    TP_ERR_VERIFY             = 0x0009,
    TP_ERR_NOT_OPEN           = 0x000A,
    TP_ERR_INVALID_PATH_ID    = 0x000B,
    TP_ERR_NULL_FUNCTION      = 0x000C,

    ABS_ERR_NOT_PARALLEL_PATH = 0x1000,
    ABS_ERR_TIMEOUT           = 0x1001,
    ABS_ERR_PARAMETER         = 0x1002,
    ABS_ERR_UNSUPPORTED       = 0x1003
}
ABS_StatusType;
```

Description:

Enumerated status code returned by the API.

Members:

| Member | Description |
|---------------------------|---|
| TP_ERR_NONE | No error. Function completed successfully. |
| TP_ERR_OTHER | General error code for errors that cannot be categorized in any other way. |
| TP_ERR_WIN_FAIL | A call to a windows API function failed. |
| TP_ERR_MEM_ALLOC | Failed to allocate needed memory. |
| TP_ERR_CONFIG | Configuration error. Most often due to a improperly configured path. |
| TP_ERR_ABORTED | The end user aborted a function. E.g. the end user did not select a path. |
| TP_ERR_MEM_SIZE | Failed to open the transport path since the desired memory window could not be supported. |
| TP_ERR_OPEN | Failed to open the transport path. |
| TP_ERR_NO_HW | The transport provider could not locate the hardware indicated. |
| TP_ERR_VERIFY | Failed to verify read/written memory. |
| TP_ERR_NOT_OPEN | Attempt to perform a function on a non-opened path that requires the path to be open. |
| TP_ERR_INVALID_PATH_ID | The indicated path id was not found in the registry. |
| TP_ERR_NULL_FUNCTION | Attempt to call a null function, i.e. the function is not supported. |
| ABS_ERR_NOT_PARALLEL_PATH | The indicated path is not a parallel path, which is required for the API |
| ABS_ERR_TIMEOUT | The function failed to complete within the indicated time frame. |
| ABS_ERR_PARAMETER | One or more of the parameters supplied in the function call is erroneous or unsupported. |
| ABS_ERR_UNSUPPORTED | The functionality is not supported by the current transport path. |

Defines

ABS_AREA_xxx

Definition:

```
#define ABS_AREA_MASK           0x07
#define ABS_AREA_FB_CTRL       0x01
#define ABS_AREA_OUT           0x02
#define ABS_AREA_IN            0x04
```

Purpose:

For use with the functions *ABS_RequestArea()* and *ABS_ReleaseArea()*.

Members:

| Member | Description |
|------------------|---|
| ABS_AREA_MASK | Used to mask out desired areas in case the low-level handshaking is performed by the application. |
| ABS_AREA_FB_CTRL | Identifies the fieldbus/control area. |
| ABS_AREA_OUT | Identifies the out area, i.e. the area to which the AnyBus module puts received data from the fieldbus for the application to read. |
| ABS_AREA_IN | Identifies the in area, i.e. the area to which the application puts data which the AnyBus module shall transmit onto the fieldbus. |

Related Information:

| Information | Page |
|--------------------------|--------------|
| <i>ABS_ReleaseArea()</i> | Appendix A-7 |
| <i>ABS_RequestArea()</i> | Appendix A-6 |

ABS_FB_xxx

Definition:

```
#define ABS_FB_PDP                0x0001
#define ABS_FB_PFMS               0x0002
#define ABS_FB_PCS                0x0003
#define ABS_FB_PPA                0x0004
#define ABS_FB_PDPV1             0x0005
#define ABS_FB_IBS               0x0010
#define ABS_FB_IBS_2MB           0x0011
#define ABS_FB_LON               0x0015
#define ABS_FB_COP               0x0020
#define ABS_FB_DEV               0x0025
#define ABS_FB_SDS               0x0030
#define ABS_FB_FIP               0x0035
#define ABS_FB_MBP               0x0040
#define ABS_FB_RTU               0x0045
#define ABS_FB_SATT              0x0050
#define ABS_FB_RIO               0x0055
#define ABS_FB_PNET              0x0060
#define ABS_FB_CNT               0x0065
#define ABS_FB_FF                0x0070
#define ABS_FB_TCP               0x0080
#define ABS_FB_IDA               0x0081
#define ABS_FB_TCPWEB            0x0082
#define ABS_FB_EIPWEB            0x0083
#define ABS_FB_CCL               0x0090
#define ABS_FB_ASI               0x0091
#define ABS_FB_VIP               0x0200
```

Purpose:

These defines are meant to be used in conjunction with the *iFieldbusType* member of the *ABS_ModuleInfoType* structure to identify the fieldbus type of the AnyBus module.

Members:

| Member | Description |
|----------------|---|
| ABS_FB_PDP | Profibus-DP |
| ABS_FB_PFMS | Profibus_FMS |
| ABS_FB_PCS | Profibus Combi Slave |
| ABS_FB_PPA | Profibus for intrinsic applications in Process Automation |
| ABS_FB_PDPV1 | Profibus-DPV1 |
| ABS_FB_IBS | InterBus |
| ABS_FB_IBS_2MB | Interbus 2 Mbit Cu/FO |
| ABS_FB_LON | Lonworks |
| ABS_FB_COP | CANopen |
| ABS_FB_DEV | DeviceNet |
| ABS_FB_SDS | SDS-Honeywell |
| ABS_FB_FIP | FIP I/O |
| ABS_FB_MBP | ModBus+ |
| ABS_FB_RTU | Modbus RTU |
| ABS_FB_SATT | Sattbus |
| ABS_FB_RIO | Remote io |
| ABS_FB_PNET | P-NET |
| ABS_FB_CNT | Controlnet |
| ABS_FB_FF | Fieldbus Foundation |
| ABS_FB_TCP | Ethernet Modbus/TCP |
| ABS_FB_IDA | Ethernet IDA |
| ABS_FB_TCPWEB | Ethernet Modbus/TCP + WEB |
| ABS_FB_EIPWEB | Ethernet IP + WEB |
| ABS_FB_CCL | CC-LINK |
| ABS_FB_ASI | AS-Interface |
| ABS_FB_VIP | Ethernet VIP |

Related Information:

| Information | Page |
|--------------------|------|
| ABS_ModuleInfo() | 4-16 |
| ABS_ModuleInfoType | 5-3 |

ABS_LED_xxx

Definition:

```
#define ABS_LED_OFF           0
#define ABS_LED_GREEN        1
#define ABS_LED_RED          2
```

Purpose:

Defines for use with the *ABS_SetApplicationLed()* function and for decoding the Led information contained in the *ABS_ModuleStatusType* structure.

Members:

| Member | Description |
|---------------|------------------------------|
| ABS_LED_OFF | Represents a turned off led. |
| ABS_LED_GREEN | Represents a green led. |
| ABS_LED_RED | Represents a red led. |

Related Information:

| Information | Page |
|-------------------------|------|
| ABS_SetApplicationLed() | 4-20 |
| ABS_ModuleStatusType | 5-4 |

ABS_MB_xxx**Definition:**

```

/*
** Bit masks for Message Information
*/

#define ABS_MB_C_R_MASK                0x4000
#define ABS_MB_ERROR_MASK              0x8000

/*
** Bit masks for message types
*/

#define ABS_MB_MESSAGE_TYPE_MASK      0x00FF
#define ABS_MB_MESSAGE_TYPE_APP       0x0001
#define ABS_MB_MESSAGE_TYPE_FB        0x0002
#define ABS_MB_MESSAGE_TYPE_EXT_MEM   0x0003
#define ABS_MB_MESSAGE_TYPE_FW        0x0004
#define ABS_MB_MESSAGE_TYPE_RESET     0x0005

/*
** Bit masks for mailbox error codes
*/

#define ABS_MB_ERROR_CODE_MASK        0x0F00
#define ABS_MB_ERROR_CODE_ID          0x0000
#define ABS_MB_ERROR_CODE_TYPE        0x0100
#define ABS_MB_ERROR_CODE_COMMAND     0x0200
#define ABS_MB_ERROR_CODE_SIZE        0x0300
#define ABS_MB_ERROR_CODE_FRAME_CNT   0x0400
#define ABS_MB_ERROR_CODE_FRAME_NBR   0x0500
#define ABS_MB_ERROR_CODE_OFFSET      0x0600
#define ABS_MB_ERROR_CODE_ADDRESS     0x0700
#define ABS_MB_ERROR_CODE_RESPONSE    0x0800
#define ABS_MB_ERROR_CODE_FLASH_CFG   0x0900

/*
** Defines for mailbox commands of application type
*/

#define ABS_MB_COMMAND_START_INIT     0x0001
#define ABS_MB_COMMAND_ANYBUS_INIT    0x0002
#define ABS_MB_COMMAND_END_INIT       0x0003
#define ABS_MB_COMMAND_SAVE_TO_FLASH  0x0004
#define ABS_MB_COMMAND_LOAD_FROM_FLASH 0x0005
#define ABS_MB_COMMAND_HW_CHK         0x0006

/*
** Defines for mailbox commands of extended memory type
*/

#define ABS_MB_COMMAND_RD_EXT_IN      0x0001
#define ABS_MB_COMMAND_WR_EXT_IN      0x0002
#define ABS_MB_COMMAND_CLR_EXT_IN     0x0003
#define ABS_MB_COMMAND_RD_EXT_OUT     0x0004

/*
** Defines for mailbox commands of reset type
*/

#define ABS_MB_COMMAND_SW_RESET       0x0001

```

Purpose:

These defines are intended to be used in conjunction with the *ABS_MailboxType* structure, and are intended for the most common use of the standardised mailboxes. For fieldbus specific mailboxes, please consult the appropriate fieldbus appendix. (Consult the AnyBus-S Parallel Design Guide for more information regarding mailbox communication.)

Members:

| Member | Description |
|--------------------------------|--|
| ABS_MB_C_R_MASK | Used to set/clear the Command/Response bit in the message information of a mailbox. |
| AB_MB_ERROR_MASK | Used to set/clear the error bit in the message information of a mailbox. |
| ABS_MB_MESSAGE_TYPE_MASK | Used to mask out the message type from the message information of a mailbox. |
| ABS_MB_MESSAGE_TYPE_APP | Identifies an application type mailbox |
| ABS_MB_MESSAGE_TYPE_FB | Identifies a fieldbus specific type mailbox. |
| ABS_MB_MESSAGE_TYPE_EXT_MEM | Identifies an extended memory type mailbox. |
| ABS_MB_MESSAGE_TYPE_FW | Identifies a firmware download type mailbox. |
| ABS_MB_MESSAGE_TYPE_RESET | Identifies a reset type mailbox. |
| ABS_MB_ERROR_CODE_MASK | Used to mask out the error code from the message information of a mailbox. |
| ABS_MB_ERROR_CODE_ID | Invalid Message ID |
| ABS_MB_ERROR_CODE_TYPE | Invalid Message Type |
| ABS_MB_ERROR_CODE_COMMAND | Invalid Command |
| ABS_MB_ERROR_CODE_SIZE | Invalid Data Size |
| ABS_MB_ERROR_CODE_FRAME_CNT | Invalid Frame Count |
| ABS_MB_ERROR_CODE_FRAME_NBR | Invalid Frame Number |
| ABS_MB_ERROR_CODE_OFFSET | Invalid Offset |
| ABS_MB_ERROR_CODE_ADDRESS | Invalid Address |
| ABS_MB_ERROR_CODE_RESPONSE | Invalid Response |
| ABS_MB_ERROR_CODE_FLASH_CFG | Flash Configuration Error |
| ABS_MB_COMMAND_START_INIT | The Start Init command to be used with <i>ABS_MB_MESSAGE_TYPE_APP</i> . |
| ABS_MB_COMMAND_ANYBUS_INIT | The AnyBus Init command to be used with <i>ABS_MB_MESSAGE_TYPE_APP</i> . |
| ABS_MB_COMMAND_END_INIT | The End Init command to be used with <i>ABS_MB_MESSAGE_TYPE_APP</i> . |
| ABS_MB_COMMAND_SAVE_TO_FLASH | The Save To Flash command to be used with <i>ABS_MB_MESSAGE_TYPE_APP</i> . |
| ABS_MB_COMMAND_LOAD_FROM_FLASH | The Load From Flash command to be used with <i>ABS_MB_MESSAGE_TYPE_APP</i> . |
| ABS_MB_COMMAND_HW_CHK | The Hardware Check command to be used with <i>ABS_MB_MESSAGE_TYPE_APP</i> . |
| ABS_MB_COMMAND_RD_EXT_IN | The Read Extended In Area command to be used with <i>ABS_MESSAGE_TYPE_EXT_MEM</i> . |
| ABS_MB_COMMAND_WR_EXT_IN | The Write Extended In Area command to be used with <i>ABS_MESSAGE_TYPE_EXT_MEM</i> . |
| ABS_MB_COMMAND_CLR_EXT_IN | The Clear Extended In Area command to be used with <i>ABS_MESSAGE_TYPE_EXT_MEM</i> . |
| ABS_MB_COMMAND_RD_EXT_OUT | The Read Extended Out Area command to be used with <i>ABS_MESSAGE_TYPE_EXT_MEM</i> . |
| ABS_MB_COMMAND_SW_RESET | The Software Reset command to be used with <i>ABS_MESSAGE_TYPE_RESET</i> . |

Related Information:

| Information | Page |
|-----------------|------|
| ABS_MailboxType | 5-1 |

ABS_MODULE_TYPE_xxx

Definition:

```
#define ABS_MODULE_TYPE_SLAVE          0x0101
#define ABS_MODULE_TYPE_MASTER        0x0201
```

Purpose:

Describes the type of AnyBus module. Intended to be used in conjunction with the *iModuleType* element of the *ABS_ModuleInfoType* structure.

Members:

| Member | Description |
|------------------------|---|
| ABS_MODULE_TYPE_SLAVE | The module will act as a slave on the network. |
| ABS_MODULE_TYPE_MASTER | The module will act as a master on the network. |

Related Information:

| Information | Page |
|--------------------|------|
| ABS_ModuleInfoType | 5-3 |

ABS_MODULE_STATUS_xxx

Definition:

```
#define ABS_MODULE_STATUS_APRS          0x0400
#define ABS_MODULE_STATUS_CD           0x0200
#define ABS_MODULE_STATUS_APFC         0x0100
#define ABS_MODULE_STATUS_FBSPU        0x0008
#define ABS_MODULE_STATUS_FBS          0x0004
#define ABS_MODULE_STATUS_FBFC         0x0002
#define ABS_MODULE_STATUS_FBRs         0x0001
```

Purpose:

Bitmasks for the module status register. For detailed information regarding the module status register, please consult the AnyBus-S Design Guide. Intended to be used in conjunction with the *iModuleStatus* element of the *ABS_ModuleStatusType* structure.

Members:

| Member | Description |
|-------------------------|--|
| ABS_MODULE_STATUS_APRS | Application stopped/running. (0 = Stopped, 1 = Running) |
| ABS_MODULE_STATUS_CD | Changed data field active. (0 = Not active, 1 = Active) |
| ABS_MODULE_STATUS_APFC | In area clear/freeze on application stopped. (0 = Clear, 1 = Freeze) |
| ABS_MODULE_STATUS_FBSPU | Parameter data handling on fieldbus offline ^a |
| ABS_MODULE_STATUS_FBS | I/O (and parameter data) handling on fieldbus offline ^a |
| ABS_MODULE_STATUS_FBRs | I/O (and parameter data) handling on fieldbus offline ^a |

a. See table below ('Status Bits')

Status Bits:

| Bit | | | Fieldbus Offline Action | |
|-------|-----|------|-------------------------|----------------|
| FBSPU | FBS | FBFC | I/O | Parameter Data |
| 0 | 0 | 0 | Clear | Clear |
| 0 | 0 | 1 | Freeze | Freeze |
| 0 | 1 | 0 | Set | Set |
| 0 | 1 | 1 | Reserved | Reserved |
| 1 | 0 | 0 | Clear | Updated |
| 1 | 0 | 1 | Freeze | Updated |
| 1 | 1 | 0 | Set | Updated |
| 1 | 1 | 1 | Reserved | Reserved |

Related Information:

| Information | Page |
|----------------------|------|
| ABS_ModuleStatusType | 5-4 |

ABS_RESET_xxx

Definition:

```
#define ABS_RESET_LOW           0
#define ABS_RESET_HIGH        1
#define ABS_RESET_MODULE      2
```

Purpose:

Intended to be used as the *bType* parameter when calling the *ABS_Reset()* function.

Members:

| Member | Description |
|------------------|---|
| ABS_RESET_LOW | Pulls the reset line low (RESET active) |
| ABS_RESET_HIGH | Pulls the reset line high (RESET inactive) |
| ABS_RESET_MODULE | First pulls the line low, and then pulls the reset line high again. Thereafter waits for AnyBus to become ready to communicate. |

Related Information:

| Information | Page |
|-------------|------|
| ABS_Reset() | 4-19 |

Advanced Functions

The functions described in this chapter can be used to access the AnyBus-S memory space directly, with or without verification. Note that when using these functions, all handshaking must be performed manually by the application in order to ensure data consistency. Consult the general AnyBus-S Design Guide for more information.

Note: The functions described in this chapter are for expert users only and requires in depth knowledge of the AnyBus-S platform. If possible, use the functions described in Chapter 4 “Functions”.

The functions are grouped by their usage according to below:

- **Advanced Read / Write Operations**

These functions are used to read and write data to the AnyBus module.

| Function | Description | Page |
|---------------------------|---|------|
| ABS_ParallelRead() | Reads data from the AnyBus memory space, w.o. verification. | A-2 |
| ABS_ParallelWrite() | Writes data to the AnyBus memory space, w.o. verification. | A-3 |
| ABS_ParallelVerifyRead() | Reads data from the AnyBus memory space, with verification. | A-4 |
| ABS_ParallelVerifyWrite() | Writes data to the AnyBus memory space, with verification. | A-5 |

- **Handshaking**

These functions are used to perform the necessary handshaking required when using the advanced read / write operations.

| Function | Description | Page |
|-------------------|----------------------------------|------|
| ABS_RequestArea() | Requests a specified memory area | A-6 |
| ABS_ReleaseArea() | Releases a specified memory area | A-7 |

Advanced Read/Write Operations

ABS_ParallelRead()

Syntax:

```
ABS_StatusType ABS_ParallelRead( void*    pxPath,
                                UINT16   iOffset,
                                UINT8*   pbData,
                                UINT16   iAmount );
```

Description:

The function reads *iAmount* bytes from *iOffset* in the AnyBus dual port memory.

Notes:

The data is read directly from the AnyBus module without handshaking and may therefore cause data inconsistency if not used properly.

Parameters:

| Parameter | Description |
|-----------|--|
| pxPath | Path identifier pointer to the dual port memory which the bytes should be read from. |
| iOffset | Offset in the dual port memory to read from. |
| pbData | Pointer indicating where the read bytes shall be stored. |
| iAmount | The amount of bytes to read. |

Usage:

```
ABS_StatusType  eStatus;
UINT8          abData[ 16 ];

/*
** Read 16 bytes from offset 0x200 in the memory
** and put it into the abData array
*/

eStatus = ABS_ParallelRead( pxPath, 0x200, abData, 16 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to read out the data
    */
}
```

Related Information:

| Information | Page |
|-------------------|--------------|
| ABS_ReleaseArea() | Appendix A-7 |
| ABS_RequestArea() | Appendix A-6 |
| ABS_StatusType | 5-5 |

ABS_ParallelWrite()

Syntax:

```
ABS_StatusType ABS_ParallelWrite( void*   pxPath,
                                  UINT16  iOffset,
                                  UINT8*   pbData,
                                  UINT16  iAmount );
```

Description:

The function writes *iAmount* bytes to *iOffset* in the AnyBus dual port memory.

Notes:

This function writes directly to the AnyBus module without handshaking procedures and may therefore cause data inconsistency if not properly used.

Parameters:

| Parameter | Description |
|-----------|---|
| pxPath | Path identifier pointer to the dual port memory which the bytes should be written to. |
| iOffset | Offset in the dual port memory to write to. |
| pbData | Pointer indicating where to fetch the bytes to write. |
| iAmount | The amount of bytes to write. |

Usage:

```
ABS_StatusType  eStatus;
UINT8          abData[ 4 ] = { 0x01, 0x02, 0x03, 0x04 };

/*
** Write 1, 2, 3 and 4 to address 0x400
*/

eStatus = ABS_ParallelWrite( pxPath, 0x400, abData, 4 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to write the data
    */
}
```

Related Information:

| Information | Page |
|-------------------|--------------|
| ABS_ReleaseArea() | Appendix A-7 |
| ABS_RequestArea() | Appendix A-6 |
| ABS_StatusType | 5-5 |

ABS_ParallelVerifyRead()

Syntax:

```
ABS_StatusType ABS_ParallelVerifyRead( void*   pxPath,
                                       UINT16  iOffset,
                                       UINT8*   pbData,
                                       UINT16  iAmount,
                                       UINT16  iMaxTries );
```

Description:

The function tries to read *iAmount* bytes from *iOffset* in the AnyBus dual port memory. It reads every byte until a matching value has been read before proceeding to the next byte. It does not try to verify the read byte more than *iMaxTries* times. This function is mainly intended for use with handshaking registers.

Notes:

This function reads directly from the AnyBus module without handshaking procedures and may therefore cause data inconsistency if not properly used.

Parameters:

| Parameter | Description |
|-----------|--|
| pxPath | Path identifier pointer to the dual port memory which the bytes should be read from. |
| iOffset | Offset in the dual port memory to read from. |
| pbData | Pointer indicating where to store the read bytes. |
| iAmount | The amount of bytes to read. |
| iMaxTries | Maximum times the function will try to re-read a byte |

Usage:

```
ABS_StatusType   eStatus;
UINT8           bData;

/*
** Read 1 byte from offset 0x7FF in the memory
** and put it into bData Verify maximum of 10 times.
*/

eStatus = ABS_ParallelVerifyRead( pxPath, 0x7FF, &bData, 1, 10 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to read out the data
    */
}
```

Related Information:

| Information | Page |
|-------------------|--------------|
| ABS_ReleaseArea() | Appendix A-7 |
| ABS_RequestArea() | Appendix A-6 |
| ABS_StatusType | 5-5 |

ABS_ParallelVerifyWrite()

Syntax:

```
ABS_StatusType ABS_ParallelVerifyWrite( void*   pxPath,
                                         UINT16  iOffset,
                                         UINT8*   pbData,
                                         UINT16  iAmount,
                                         UINT16  iMaxTries );
```

Description:

The function tries to write *iAmount* bytes to *iOffset* in the AnyBus dual port memory. It writes and then reads to verify every byte until a matching value has been read before proceeding to the next byte. It does not try more than *iMaxTries* times. This function is mainly intended for use with handshaking registers.

Notes:

This function reads directly from the AnyBus module without handshaking procedures and may therefore cause data inconsistency if not properly used.

Parameters:

| Parameter | Description |
|-----------|---|
| pxPath | Path identifier pointer to the dual port memory which the bytes should be written to. |
| iOffset | Offset in the dual port memory to write from. |
| pbData | Pointer indicating where to fetch the bytes to be written. |
| iAmount | The amount of bytes to write. |
| iMaxTries | Maximum times the function will try to write and verify a byte |

Usage:

```
ABS_StatusType   eStatus;
UINT8            bData;

/*
** Write 0x14 to address 0x7FE and verify a maximum of ten times
**/

bData = 0x14;
eStatus = ABS_ParallelVerifyWrite( pxPath, 0x7FE, &bData, 1, 10 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to write the data
    **/
}
```

Related Information:

| Information | Page |
|-------------------|--------------|
| ABS_ReleaseArea() | Appendix A-7 |
| ABS_RequestArea() | Appendix A-6 |
| ABS_StatusType | 5-5 |

Handshaking

ABS_RequestArea()

Syntax:

```
ABS_StatusType ABS_RequestArea( void*    pxPath,
                                UINT8    bArea,
                                UINT16   iTimeout );
```

Description:

This function handshakes with the AnyBus module in order to retrieve the specified area(s) and thus give the application ownership of the area.

Notes:

There are functions that allow the application to request, read/write and release the area. This function is only intended for special applications that require a higher level of control than that. When the area is no longer needed by the application it should be released using *ABS_ReleaseArea()*.

Parameters:

| Parameters | Description |
|------------|---|
| pxPath | Path identifier pointer to the AnyBus module whose area(s) shall be retrieved. |
| bArea | Bitmask indicating the area(s) to request. See also 6-1 "ABS_AREA_xxx" |
| iTimeout | The maximum time the application is willing to wait for access to the area to be granted. |

Usage:

```
ABS_StatusType  eStatus;

/*
** Request the out area
*/

eStatus = ABS_RequestArea( pxPath, ABS_AREA_OUT, 20 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to get access to the area
    */
}
```

Related Information:

| Information | Page |
|-------------------|--------------|
| ABS_ReleaseArea() | Appendix A-7 |
| ABS_StatusType | 5-5 |
| ABS_AREA_xxx | 6-1 |

ABS_ReleaseArea()

Syntax:

```
ABS_StatusType ABS_ReleaseArea( void*   pxPath,
                               UINT8   bArea,
                               UINT16  iTimeout );
```

Description:

This function handshakes with the AnyBus module in order to release the specified area(s) and thus return ownership of it/them to the AnyBus module.

Notes:

There are functions that allow the application to request, read/write and release the area. This function is only intended for special applications that require a higher level of control than that. The area to be released should have been successfully requested by prior call to *ABS_RequestArea()*.

After the call a new call to *ABS_RequestArea()* should be performed prior to accessing the area(s) again.

Parameters:

| Parameters | Description |
|------------|--|
| pxPath | Path identifier pointer to the AnyBus module whose area(s) shall be released. |
| bArea | Bitmask indicating the area(s) to be released. See also 6-1 "ABS_AREA_xxx" |
| iTimeout | The maximum time the application is willing to wait for the area to be released. |

Usage:

```
ABS_StatusType  eStatus;

/*
** Release the previously successfully requested out area
*/

eStatus = ABS_ReleaseArea( pxPath, ABS_AREA_OUT, 20 );

if( eStatus != TP_ERR_NONE )
{
    /*
    ** Failed to release the area
    */
}
```

Related Information:

| Information | Page |
|-------------------|--------------|
| ABS_RequestArea() | Appendix A-6 |
| ABS_StatusType | 5-5 |
| ABS_AREA_xxx | 6-1 |

Troubleshooting

As a general rule, it is recommended to read the AnyBus-S and AnyBus-M Design Guides thoroughly before contacting technical support. Answers to some common questions are listed below.

How do I install the library files in my compiler?

- This is not the scope of this document. Consult the compiler/interpreter documentation for more information.

Where do I find library files for Delphi and Visual Basic?

- It is possible to access the API from Delphi and Visual Basic, however no library files are supplied by HMS at the time of writing.

Sometimes data is byte-swapped, some times it is not. How do I know when?

- Functions performing word access will always byteswap. However, functions performing byte-wise storage cannot determine whether the intention is byte or word access, and will therefore never perform any byte swapping. Keep this in mind when mixing byte and word oriented functions. (See 3-1 “Byte Swapping” for more information.)

AnyBus-S module does not exchange data.

- The module has to be initialised in order to be able to exchange data. Initialisation is performed using mailbox commands and is described in the general AnyBus-S Design Guide.

The API refuses to perform any read/write operations at all.

- Before any read / write operations can be performed, a transport path to the AnyBus-S module has to be established. (See 3-2 “Path Management” for more information.)

Input/Output definition seems to differ between AnyBus-S master and slave.

- This is correct. Consult the AnyBus-M and AnyBus-S Design Guides for more information.

ABS_Reset() returns an error.

- Some interfaces does not support a reset line. In these cases, this is normal.

API functions takes ages to respond

- Decrease the timeout value. See 3-1 “Timeout Values” for more information.

API functions times out randomly

- Increase the timeout value. See 3-1 “Timeout Values” for more information.

