



# **Teradata Tools and Utilities Access Modules Programmer Guide**

**Teradata Tools and Utilities, Release 07.01.00**

B035-2424-093A  
September 2003

---

The product described in this book is a licensed product of NCR Corporation.

BYNET is an NCR trademark registered in the U.S. Patent and Trademark Office.

CICS, CICS/400, CICS/600, CICS/ESA, CICS/MVS, CICS/PLEX, CICS/VIEW, CICS/VSE, DB2, DFSMS/MVS, DFSMS/VM, IBM, NQS/MVS, OPERATING SYSTEM/2, OS/2, PS/2, MVS, QMS, RACF, SQL/400, VM/ESA, VTAM, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the U. S. and other countries.

DEC, DECNET, MICROVAX, VAX and VMS are registered trademarks of Digital Equipment Corporation.

HEWLETT-PACKARD, HP, HP BRIO, HP BRIO PC, and HP-UX are registered trademarks of Hewlett-Packard Co.

KBMS is a trademark of Trinzic Corporation.

INTERTEST is a registered trademark of Computer Associates International, Inc.

MICROSOFT, MS-DOS, MSN, The Microsoft Network, MULTIPLAN, SQLWINDOWS, WIN32, WINDOWS, and WINDOWS NT are trademarks or registered trademarks of Microsoft Corporation.

SAS, SAS/C, SAS/CALC, SAS/CONNECT, and SAS/CPE are registered trademarks of SAS Institute Inc.

SOLARIS, SPARC, SUN and SUN OS are trademarks of Sun Microsystems, Inc.

TCP/IP protocol is a United States Department of Defense Standard ARPANET protocol.

TERADATA is a registered trademark of NCR International, Inc.

UNICODE is a trademark of Unicode, Inc.

UNIX is a registered trademark of The Open Group.

X and X/OPEN are registered trademarks of X/Open Company Limited.

YNET is a trademark of NCR Corporation.

**THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN "AS-IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. IN NO EVENT WILL NCR CORPORATION (NCR) BE LIABLE FOR ANY INDIRECT, DIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS OR LOST SAVINGS, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.**

The information contained in this document may contain references or cross references to features, functions, products, or services that are not announced or available in your country. Such references do not imply that NCR intends to announce such features, functions, products, or services in your country. Please consult your local NCR representative for those features, functions, products, or services available in your country.

Information contained in this document may contain technical inaccuracies or typographical errors. Information may be changed or updated without notice. NCR may also make improvements or changes in the products or services described in this information at any time without notice.

To maintain the quality of our products and services, we would like your comments on the accuracy, clarity, organization, and value of this document. Please e-mail: [teradata-books@lists.ncr.com](mailto:teradata-books@lists.ncr.com)

or write:

Information Engineering  
NCR Corporation  
17095 Via Del Campo  
San Diego, California 92127-1711  
U.S.A.

Any comments or materials (collectively referred to as "Feedback") sent to NCR will be deemed non-confidential. NCR will have no obligation of any kind with respect to Feedback and will be free to use, reproduce, disclose, exhibit, display, transform, create derivative works of and distribute the Feedback and derivative works thereof without limitation on a royalty-free basis. Further, NCR will be free to use any ideas, concepts, know-how or techniques contained in such Feedback for any purpose whatsoever, including developing, manufacturing, or marketing products or services incorporating Feedback.

**Copyright © 1999-2003, NCR Corporation  
All Rights Reserved**

# Preface

## Purpose

The Teradata<sup>®</sup> Tools and Utilities are a group of products designed to work with the Teradata Database. This book describes the access module component of the Teradata Tools and Utilities infrastructure that links client utilities to external data source/destination storage devices. The book explains the standard software functions and protocols for providing a block-level I/O interface between the external devices and the client utility Data Connector Application Program Interface (API).

## Supported Releases

This book supports the following releases:

- Teradata V2R5.1
- Teradata Tools and Utilities 07.01.00

## Changes to This Book

The following changes were made to this book to support the current release:

Date	Description
December 2002	<ul style="list-style-type: none"><li>• Added supported data formats for Teradata Warehouse Builder DataConnector operator.</li><li>• Updated header file examples in Appendix A.</li></ul>
September 2003	<ul style="list-style-type: none"><li>• Listed attributes that are sent to access modules from the Teradata Warehouse Builder DataConnector operator and the standalone DataConnector.</li></ul>

## Audience

This book is intended for system and application programmers.

## Prerequisites

This book is a reference document. There is no prerequisite reading, but you should be familiar with computer I/O technology, relational database management systems, and utilities that load and retrieve data.

## Product Related Publications

The publication that supports the Access Module products is listed below. In the following table, the *mmyx* represents the publication date, where *mm* is the month, *y* is the last digit of the year, and *x* is an internal publication code.

---

B035-2425- <i>mmyx</i>	<i>Teradata Tools and Utilities Access Modules Reference</i>
------------------------	--

---

## Teradata Tools and Utilities Release Definition

The *Teradata Tools and Utilities Release Definition* gives you any additional information received late in the release process that may not have been included in this document.

You can also find a list of:

- Operating systems and Teradata Database versions that the Teradata Tools and Utilities release are certified to work with
- Product release version numbers
- Documentation that supports the current release
- Training and support centers

## Technical Information on the Web

For technical support, additional information, or the latest versions of Teradata publications, you may access online information through the following NCR Web sites:

<a href="http://www.info.ncr.com/">http://www.info.ncr.com/</a>	<p>A direct link to NCR Information Products Publishing library where you can view, download, or order technical documentation and CD-ROMs.</p> <p>You can also find a Documentation List link for each Teradata Tools and Utilities release. The Documentation List identifies all publications released in support of the current Teradata Database and the Teradata Tools and Utilities release.</p>
<a href="http://www.teradata.com">Teradata.com</a>	<p>The Teradata home page provides links to numerous sources of information about Teradata. Among the links provided are:</p> <ul style="list-style-type: none"> <li>• <b>Executive Center</b>—Executive reports, case studies of customer experiences with Teradata, and thought leadership</li> <li>• <b>Tech Center</b>—Technical information, solutions, and expert advice</li> <li>• <b>Media Center</b>—Press releases, mentions and media resources</li> <li>• <b>Teradata User Groups</b></li> </ul>

## List of Acronyms

The following acronyms are used in this book.

API	Application Program Interface
DBS	Database System
I/O	Input/Output



# Contents

---

## Preface

Purpose .....	i
Supported Releases .....	i
Changes to This Book .....	i
Audience .....	i
Prerequisites .....	i
Product Related Publications.....	ii
Teradata Tools and Utilities Release Definition.....	ii
Technical Information on the Web.....	ii
List of Acronyms .....	iii

---

## Chapter 1:

### Overview

Access Modules .....	1-2
What is an Access Module? .....	1-2
The Data Connector API.....	1-2
Record Formats.....	1-2
Defining the DataConnector Operator and Specifying Access Modules.....	1-6
Defining the DataConnector Operator .....	1-6

---

## Chapter 2:

### Using Access Modules

Functional Description .....	2-2
The Access Module Main Function .....	2-2
The Access Module Interface.....	2-2
Attribute Exchange Functions .....	2-3
Definition .....	2-3
Get Functions .....	2-3
Put Functions .....	2-4
Using the Attributes Provided by Client Utilities .....	2-4

---

**Chapter 3:**  
**Access Module Functions**

The General Function-Independent Structure ..... 3-2

Initialization ..... 3-5

Identification ..... 3-8

File Open..... 3-10

File Close..... 3-13

File Read ..... 3-15

File Write..... 3-17

File Get Position..... 3-19

File Set Position..... 3-21

Put Attribute..... 3-23

Get Attribute ..... 3-25

Shutdown..... 3-27

---

**Appendix A:**  
**Access Module Header File Examples**

Access Modules ..... A-2

    pmdcomt.h..... A-2

Data Connector APIs..... A-6

    pmddamt.h ..... A-6

Teradata Warehouse Builder DataConnector Operator ..... A-11

    pmdcomt.h..... A-11

Teradata Warehouse Builder DataConnector Operator Module ..... A-14

    pmddamt.h ..... A-14

---

**Appendix B:**  
**Access Module Examples**

Sample Program ..... B-2

    EGaxsm.c..... B-2

Sample Makefile ..... B-11

---

**Index..... Index-1**



# List of Figures

Figure 1-1	Access Module Linkage.....	1-2
------------	----------------------------	-----





## List of Tables

Table 1-1	Supported Record Formats for Client Utilities .....	1-3
Table 1-2	Supported Record Formats for the Teradata Warehouse Builder DataConnector Operator .....	1-4
Table 3-1	Rectype Function Requests.....	3-3
Table 3-2	Typical Interface Return Codes .....	3-4



  
**Chapter 1:**

# Overview

This chapter provides a general overview of access modules, including:

- The access module interface with Teradata client utilities through the Data Connector Application Program Interface (API) module
- The supported record formats for read and write operations

# Access Modules

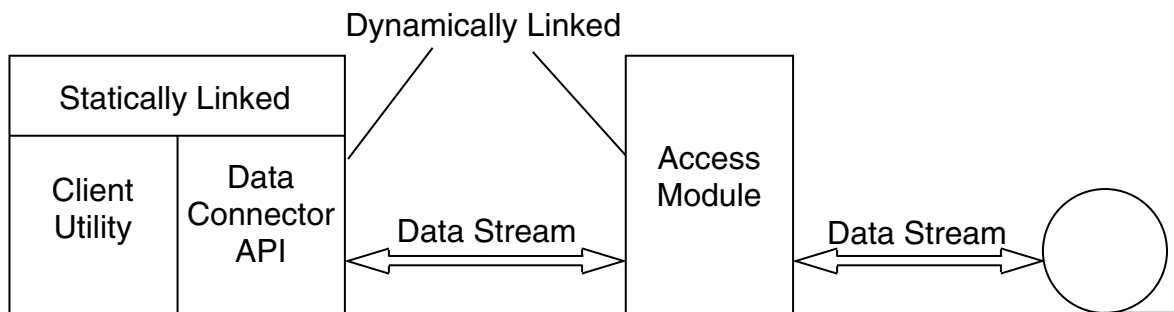
## What is an Access Module?

An access module is a software component of the Teradata client utility infrastructure that provides a block-level I/O interface to data residing on a specific external data storage device.

Each access module is tailored to a specific type of external data storage device, and can be dynamically linked to one or more Teradata client utilities by the Data Connector Application Program Interface (API), as shown in Figure 1-1.

**Note:** If you have a Teradata Warehouse Builder version of an access module, the term *Data Connector* refers to the Teradata Warehouse Builder DataConnector operator.

Figure 1-1 Access Module Linkage



KO01A006

## The Data Connector API

The Data Connector API is a software component of each Teradata client utility infrastructure that provides a block-level I/O interface to one or more access modules.

Each Data Connector API supports the logical record level I/O of the Teradata client utility, and is statically linked to the utility as shown in [Figure 1-1](#).

## Record Formats

The Data Connector API blocks data for write operations and unblocks data for read operations according to the record format specified by the client utility or Teradata Warehouse Builder DataConnector operator. The supported formats for client utilities are shown in [Table 1-1](#); formats for the Teradata Warehouse Builder DataConnector operator are in [Table 1-2](#).

The actual format that is used is determined at runtime by the user. When developing an access module, remember that the user may need to know:

- The format of the source data for both read and write operations
- That the access module supports the required format and operation

Always provide user documentation that clearly identifies the record formats and operations (read, write, or both) that are supported by the access module.

For . . . operations	The data flow is . . .
read	from the access module to the client Data Connector API.  The Data Connector API expects, but does not require the access module to provide data in blocks that are comprised of one or more logical records.
write	from the Data Connector API to the access module.  The Data Connector API in most cases provides data in blocks that are comprised of one or more logical records.

Table 1-1 Supported Record Formats for Client Utilities

Format	Description
pmIDFtext	Records are separated by end-of-record markers.
pmIDFBin1	Data is prefixed by a two-byte record length. This length does not include the two bytes used for the record length.
pmIDFBin1Plus	Same as pmIDFBin1 except that each record is followed by an end-of-record marker.
pmIDFBin2	Same as pmIDFBin1 except that the 2 bytes containing the record length include the 2 bytes used for the record length itself.
pmIDFBin2Plus	Same as pmIDFBin2 except that each record is followed by an end-of-record marker.
pmIDFnone	Data has no internal formatting. In this case, it is the responsibility of the utility to determine record separation, if any.

**Note:**

- The mnemonics, such as pmIDFBin1, are used for clarity and reference purposes only.
- End-of-record markers are one-byte fields containing either decimal '10' (x'0a') or decimal '13' (x'0d').

Table 1-2 Supported Record Formats for the Teradata Warehouse Builder DataConnector Operator

Format	Description
Binary	<p>Each record contains a 2-byte integer data length, <i>n</i>, followed by <i>n</i> bytes of data.</p> <p><b>Example</b></p> <p>A record containing 6 bytes of text "abcdef" would be:</p> <ul style="list-style-type: none"> <li>• Hexadecimal: x'00 06 61 62 63 64 65 66'</li> <li>• Character: "..abcdef"</li> </ul>
Text	<p>Each record contains character data only. Records are separated by an end-of-record (EOR) marker.</p> <p><b>Note:</b> The EOR marker can be either a single-byte line feed (X'0A') or a double-byte carriage-return/line feed pair (X'0D0A'). The first EOR marker in the data marks the end of the first record; the end of each remaining record should also be marked with an EOR marker.</p> <p><b>Example</b></p> <p>A record containing the text "acdc" would be:</p> <ul style="list-style-type: none"> <li>• Hexadecimal: x'61 63 64 63 0a'</li> <li>• Character: "abcd."</li> </ul>
Delimited	<p>Each record is in text format and contains fields (columns) separated by a delimiter character. This delimiter character is provided to the Teradata Warehouse Builder DataConnector operator through its TextDelimiter attribute. If not provided, the TextDelimiter attribute defaults to the pipe character (' ').</p> <p><b>Note</b> The final delimiter (of the last column) is optional.</p> <p><b>Example</b></p> <p>Using a colon as the delimiter (TextDelimiter = ':'), a record with two fields, text "1234" and "abcd", would be:</p> <ul style="list-style-type: none"> <li>• Hexadecimal: x'31 32 33 34 3a 61 62 63 64 3a 0a'</li> <li>• Character: "1234:abcd:."</li> </ul>

Table 1-2 (Continued) Supported Record Formats for the Teradata Warehouse Builder DataConnector Operator

Format	Description
Formatted	<p>Each record is in a format traditionally known as FastLoad or Teradata format. The data is prefixed with the data length (as with binary format) and followed by an end-of-record (EOR) marker (as with text format).</p> <p><b>Example</b></p> <p>A record containing 6 bytes of text "abcdef" would be:</p> <ul style="list-style-type: none"> <li>Hexadecimal: x'00 06 61 62 63 64 65 66 0a'</li> <li>Character: "..abcdef."</li> </ul>
Unformatted	<p>The data does not conform to any predefined format. The data is entirely described by the specified Teradata Warehouse Builder schema.</p> <p><b>Example</b></p> <pre> DEFINE SCHEMA PRODUCT_SOURCE_SCHEMA DESCRIPTION 'PRODUCT INFORMATION SCHEMA' (   COL1 INTEGER,   COL2 CHAR(4),   COL3 VARCHAR(8) ); </pre> <p>A record containing the data: integer 20, char "1234", varchar "abcde" would be;</p> <pre>x'00 00 00 14 31 32 33 34 00 05 61 62 63 64 65'</pre> <p>where</p> <p>x'00 00 00 14' is the integer 20,</p> <p>x'31 32 33 34' is the fixed length character "1234"</p> <p>x'00 05' is the length of the varchar field and</p> <p>x'61 62 63 64 65' is the varchar character data "abcde"</p>

# Defining the DataConnector Operator and Specifying Access Modules

Teradata Warehouse Builder operators, which emulate functions such as FastExport and FastLoad, interface with access modules through the DataConnector operator. You specify an access module to the DataConnector operator when you are defining the DataConnector operator.

## Defining the DataConnector Operator

To define the DataConnector operator in the Teradata Warehouse Builder job script:

- Include the following program text within the DataConnector DEFINE OPERATOR ATTRIBUTES clause (in addition to the other attributes you are defining):

```
(
  VARCHAR AccessModuleName,
  VARCHAR AccessModuleInitStr
)
```

The two operator attributes in the previous code sample specify an access module:

Attribute Name	Type	Description
AccessModuleName	VARCHAR	Specifies the file name of the dynamically loadable module that provides the access module software.
AccessModuleInitStr	VARCHAR	Specifies the access module initialization string, which is a list of operational parameters that you can specify for the access module. The contents of the initialization string are determined by the requirements of the specified access module.

- Define the access module attributes within any LOAD or SELECT clause in which you are using the DataConnector operator. For example:

```
SELECT * FROM OPERATOR
  ( DC_FILE_READER() [1]
  ATTR
  (
    PrivateLogName = 'dcR.log',
    FileName = './dummy',
    OpenMode = 'Read',
    Format = 'Text',
    IndicatorMode = 'No',
    AccessModuleName = 'libmqz.so',
    AccessModuleInitStr = '-qmgr SYSQ1 -qnm MyQ -TRCL 2
MQTRCE.txt'
  )
  );
```

For the appropriate value to assign to the Format attribute, refer to specific access module documentation.



  
**Chapter 2:**

# Using Access Modules

This chapter describes the functional aspects of the access module interface with the data connector component of the Teradata client utility (that is, Data Connector APIs or Teradata Warehouse Builder DataConnector operator).

---

# Functional Description

## The Access Module Main Function

Each access module must be packaged as a single main function that will be called by the data connector component of the Teradata client utility for all device-specific I/O functions. The structure for the main function is:

```
void PIDMMain ( pmiCmdBlock_t *Opts, void *OptParms )
```

[Appendix B](#) provides a sample program that supports the access module interface.

## The Access Module Interface

The entire access module interface is supported by two parameters—there are no other elements such as global or environmental variables. Each parameter is a structure pointer, and the structures provide the communication between the access module and the Teradata client utility:

- The first parameter specifies a general, function-*independent* structure that contains the common fields required for all access module functions.
- The second parameter specifies one of 11 function-*dependent* structures that contains the unique fields required for a specific access module function:
  - [Initialization](#)
  - [Identification](#)
  - [File Open](#)
  - [File Close](#)
  - [File Read](#)
  - [File Write](#)
  - [File Get Position](#)
  - [File Set Position](#)
  - [Put Attribute](#)
  - [Get Attribute](#)
  - [Shutdown](#)

The second parameter is passed as a void.

[Chapter 3](#) provides a detailed description of each access module function, as well as the corresponding function-dependent structure.

# Attribute Exchange Functions

## Definition

Four of the function-dependent access module structures provide a method of exchanging informational values between the access module and the Teradata client utility:

This function . . .	Transfers an attribute value from the . . .	To the . . .
Identification Get attribute	access module	client utility.
Put attribute	client utility	access module.

**Note:** The attribute exchange functions provide only an *exchange* of information—no *control* is implied. The use and disposition of the requested information is the responsibility of the destination module.

These functions typically, for example, exchange identification of the:

- Teradata Database table that is currently being addressed from the client utility to the access module
- VOLSER of the currently mounted media from the access module to the client utility

## Get Functions

In response to a get attribute request, the access module must respond with either:

- The value of the requested information
- A return code indicating that the attribute name was not recognized

This is a passive function, in that the access module can only accumulate and maintain the value of the requested information. It can only send the value to the client utility in response to a specific request. If, for example, the access module is supposed to provide the client utility with the value of several attributes that describe its state of operation, the access module cannot actively send this information to the client utility. Examples of these attributes are:

- DeviceState
- MediaMounted
- VolID
- DegreesC

The access module must wait for the utility to request the information with a get request. When the access module receives the get request, it responds with the value for the requested attribute.

## Put Functions

In response to a put attribute request, which includes a value for an attribute that has been previously defined by the client utility, the access module must respond with a return code indicating whether the attribute name was recognized or not.

This is a passive function, in that the access module can only receive the value of the provided information. It cannot actively retrieve the information from the client utility.

If, for example, the client utility is supposed to provide the access module with the value of several attributes that describe its state of operation, such as:

- DBSVersion
- ClientState
- CurrDBS
- CurrTable

the client utility can perform a put attribute function whenever one of these attributes is first defined or changed. The access module receives each put attribute request and responds each time, indicating whether the attribute name was recognized or not.

**Note:** If the access module requires a given attribute to be defined, and that attribute has not been defined by a put attribute function from the client utility before the time it is needed, then the access module may fail, and respond with the appropriate return code and an error message.

## Using the Attributes Provided by Client Utilities

The next table has the attributes that client utilities provide to the access modules. `CHARSET_NAME` and `CHARSET_NUMBER`, for example, are part of the session character set that is passed by `pmAPutAttr()` from the client utility to the access module:

Name of Attribute	Provided by:	Attribute Value
CHARSET_NAME	FastExport, FastLoad, MultiLoad, and TPump job scripts	Character set name. This attribute is optional.
CHARSET_NUMBER	FastExport, FastLoad, MultiLoad, and TPump job scripts	1-byte id of the character set. This attribute is optional.
TBR_OP_HANDLE	Teradata Warehouse Builder DataConnector operator	<p>Pointer to Teradata Warehouse Builder operator handle. The pointer is used to communicate with Teradata Warehouse Builder functions.</p> <p>The Teradata Warehouse Builder DataConnector operator always sends this attribute, although the access module does not have to use it.</p>



## Chapter 3:

# Access Module Functions

This chapter describes the functionality required from an access module, as well as the supporting data structures. This information is necessary if you are going to write an access module to support an I/O subsystem that is not currently supported by the Teradata system. For example, you might want to write a module to access a CD-ROM burner or an automated satellite up/downlink.

All of the program examples and descriptions use the 'C' programming language.

## Programming Considerations

When writing an access module:

- All functions are invoked with support of two parameters:
  - The general function-independent specifications
  - The function-dependent specifications
- Use an initialization function first, before any other.
- Specify all buffer and record lengths as multiples of the *sizeof(pmChar\_t)* value.
- Free all buffer areas that were created by the access module when they are no longer needed.

## UNIX Signals

Signals are predefined messages sent between two UNIX<sup>®</sup> processes to communicate the occurrence of unexpected external events, or exceptions. You cannot incorporate the UNIX signals that client utilities use into an access module or routine for the utility. Doing so will cause an error.

Refer to the client utility documentation for a list of the UNIX signals the utility uses (for example, one of the signals FastLoad uses is SIGUSR1).

# The General Function-Independent Structure

## Function

The general function-independent structure identifies the requested function and provides the specifications that are required by all access module function calls.

## Structure

For detailed examples, refer to the `pmdcomt.h` and `pmddamt.h` header files shown in [Appendix A](#).

```
typedef struct _pmiCmdBlock
{
    char          EyeCatcher [pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32      StructLength;                /* Length of this structure */
    pmUInt32      Reqtype;                     /* function request indicator */
    pmReturnType  Retcode;                     /* identical to function return value */
    pmTrceLvl_t  TraceLvl;                    /* Requested diagnostic trace level */
                                                    /* Descriptive error text */
    pmNameBuf_t  ErrMsg;                       /* when Retcode is pmrcFailure */
} pmiCmdBlock_t;
```

where,

Parameter ...	Is the ... field	That contains the ...
EyeCatcher	input	structure description string, such as <code>pmCmdBlock</code> .
StructLength	input	total structure length, including the <code>EyeCatcher</code> string.
Reqtype	input	specific function request, as listed in <a href="#">Table 3-1</a> .
Retcode	output	value assigned by the access module function as one of those defined in the <code>pmdcomt.h</code> header file.  Although additional function-specific return codes are defined with each function, typical interface return codes are listed in <a href="#">Table 3-2</a> .
TraceLvl	input	default trace level to be associated with all functions related to this operation.

Parameter . . .	Is the . . . field	That contains the . . .
ErrMsg	output	<p>two-field value assigned by the access module function when Retcode is set to pmrcFailure. These fields provide the:</p> <ul style="list-style-type: none"> <li>• Descriptive error text of up to 32,000 bytes, maximum</li> <li>• Length of the text</li> </ul> <p>Whenever an error condition is detected, the access module must map the error to one of the error codes defined in the pmdcomt.h header file. If a well-defined match is not found, the access module must return a value of pmrcFailure and provide an error description string in this ErrMsg field.</p> <p>The client utility then presents the error message to the user.</p>

Table 3-1 Reqtype Function Requests

Reqtype	Description
pmiPIDMOptInit	Access module initialization.
pmiPIDMOptOpen	File open.
pmiPIDMOptClose	File close.
pmiPIDMOptCloseR	File close and release media.
pmiPIDMOptRead	File read.
pmiPIDMOptWrite	File write.
pmiPIDMOptGetPos	Get position of currently open file.
pmiPIDMOptSetPos	Set position of currently open file.
pmiPIDMOptShut	Access module shutdown (cleanup).
pmiPIDMOptID	Access module identification.
pmiPIDMOptGetA_A	Get (obtain) an access module attribute.
pmiPIDMOptGetF_A	Get (obtain) a file attribute.
pmiPIDMOptPutA_A	Put (provide) an access module attribute.
pmiPIDMOptPutF_A	Put (provide) a file attribute.

Chapter 3: Access Module Functions  
**The General Function-Independent Structure**

Table 3-2 Typical Interface Return Codes

Return Code	Description
pmrcOK	No error conditions encountered.
pmrcUnsupported	The function requested by Rectype is not supported.
pmrcBadTraceLvl	The value of TraceLvl is out of range.
pmrcBadParm	One (or more) of the supplied parameters is invalid.
pmrcAPInotReady	The access module has not been initialized.
pmrcAllocErr	A memory allocation error occurred.
pmrcFailure	General failure return code. (See the description of the ErrMsg parameter.)

# Initialization

## Function

The initialization function initializes the access module.

## Structure

```
typedef struct _pmiInit
{
    char      EyeCatcher [pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32 StructLength;                /* Length of this structure */
    void      *PIData;                    /* Access module Internal Data Pointer */
    pmUInt16 InterfaceVerNo;              /* pmdcomt.h version identifier */
    pmUInt16 InterfaceVerNoD;            /* pmddamt.h version identifier */
    pmUInt32 ClientIDL;                   /* Length of ClientID field */
    pmUInt32 InitStrL;                    /* Length of InitStr field */
                                           /* Client Utility */
    char      ClientID [pmMAX_CLIENT_ID+1]; /* identifier */
                                           /* Access module */
    char      InitStr [pmiMAX_INIT_STR_LEN+1]; /* initialization field */
} pmiInit_t;
```

where,

Parameter ...	Is the ... field	That specifies ...
EyeCatcher	input	the structure description string, such as pmInitParms.
StructLength	input	the total structure length, including the EyeCatcher string.
PIData	output	a handle/pointer to internal access module data.  There are no restrictions on the PIData value, and the assigned value will be returned in a subsequent call to the access module.
InterfaceVerNo	input	the identifying value for the expected version of the general access module interface header file, pmdcomt.h.  This value should match the pmInterfaceVersion value defined in pmdcomt.h. If the match fails, the access module should set Retcode to pmrcBadVer and return.

Parameter . . .	Is the . . . field	That specifies . . .
InterfaceVerNoD	input	the identifying value for the expected version of the device-dependent access module interface header file, pmddamt.h.  This value should match the pmiInterfaceVersion value defined in pmddamt.h. If the match fails, the access module should set Retcode to pmrcBadVer and return.
ClientIDL	input	the length of ClientID.
InitStrL	input	the length of InitStr.
ClientID	input	an identifier of the using program, such as FastLoad.
InitStr	input	a string of access module-specific initialization information that is meaningful only to the access module.  The customer provides the initialization string, by way of the client utility, which passes the string unaltered to the access module.

## Return Codes

The following initialization return codes supplement those listed in [Table 3-2](#).

Return code . . .	Indicates . . .
pmrcBadVer	an invalid parameter value for InterfaceVerNo or InterfaceVerNoD.
pmrcRedundant	that the access module has already been initialized.

## Usage Notes

The topics in the following table describe the things you should consider when specifying the initialization parameters.

Topic	Usage Notes
Initialization Function Requirements	Access module initialization is required when the Reqtype field of the pmiCmdBlock_t structure is pmiPIDMOptInit.  This function must be called at least one time.

Topic	Usage Notes
Supporting Multiple Iterations	<p>When supporting more than one iteration, use the PIData field to distinguish between the:</p> <ul style="list-style-type: none"> <li>• Different iterations</li> <li>• Files opened by each iteration</li> </ul> <p>To distinguish between different iterations, use a different PIData value with each initialization request.</p> <p>To distinguish between the files opened by each iteration, use the same PIData value with each file open request.</p>

---

# Identification

## Function

The identification function identifies the access module and the name and version number of all internal modules.

## Structure

```
#define pmMAX_VER_STR_LEN 31
typedef struct _pmVerLst_t
{
    char          ModuleName [pmMAX_VER_STR_LEN+1];
    char          ModuleVers [pmMAX_VER_STR_LEN+1];
    struct _pmVerLst_t *Next;
} pmVerLst_t;
typedef struct _pmiID
{
    char          EyeCatcher [pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32     StructLength;                /* Length of this structure */
    pmVerLst_t   VerIDList;
} pmiID_t;
```

where,

Parameter ...	Is the ... field	That specifies the ...
EyeCatcher	input	structure description string, such as pmIDParms.
StructLength	input	total structure length, including the EyeCatcher string.
VerIDList	output	first of a singly linked list of module identifications.

## Return Codes

The identification return codes are as listed in [Table 3-2](#).

## Usage Notes

The topics in the following table describe the things you should consider when specifying the identification function parameters.

Topic	Usage Notes
Identification Function Requirements	Access module identification is required when the Rectype field of the pmiCmdBlock_t structure is pmiPIDMOptId.
Modules to List	Include all of the source modules in the list.

---

# File Open

## Function

The file open function opens a file for subsequent read and write operations.

**Definition of *file*:** The term *file* refers to a data source or destination on any sort of data storage device, such as magnetic tape, disk, CD-ROM, another database, and so forth, that the access module supports.

**Definition of *read* and *write*:** The terms *read* and *write* refer to data moving from (read) and to (write) the data source/destination.

## Structure

```
typedef struct _pmiOpen
{
    char            EyeCatcher [pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32        StructLength;                /* Length of this structure */
    void            *PIData;                      /* Access module Internal Data Pointer */
    pmOpenMode_t    OpenMode;                    /* provided to Access Module */
    pmUInt16        BlkHeaderLen;                /* defined by Access Module */
    pmUInt32        BlkSize;                     /* defined by Access Module */
                                                         /* File Internal Data */
    void            *FIData;                      /* (returned/used by Access Module) */
                                                         /* File (data stream) supplied to Access Module */
    pmUInt16        FileNameL;                   /* Name length */
    char            FileName [pmiMAX_DDNAME_LEN+1]; /* Name */
} pmiOpen_t;
```

where,

Parameter ...	Is the ... field	That specifies the ...
EyeCatcher	input	structure description string, such as pmOpenParms.
StructLength	input	total structure length, including the EyeCatcher string.
PIData	input	value provided by the access module in response to a previous initialization request call.  This value identifies the instance of the access module opening the file.

Parameter ...	Is the ... field	That specifies the ...
OpenMode	input	value indicating the intended use of the file to be opened. The following values are supported. <ul style="list-style-type: none"> <li>pmOpenModeR—Opens the file for read-only access.</li> <li>pmOpenModeW—Opens the file for write-only access.</li> <li>pmOpenModeRW—Opens the file for write-only access, appending to the end of the file.</li> </ul>
BlkHeaderLen	output	number of device-dependent bytes that prefix the user data in each block.
BlkSize	output	size, in bytes, of the data blocks to be either read from or written to the file.
FIData	output	handle/pointer to internal access module data.  There are no restrictions on the FIData value, and the assigned value will be returned in a subsequent call to the access module.
FileNameL	input	length of FileName parameter.
FileName	input	name of the file to be opened.

## Return Codes

The following file open return codes supplement those listed in [Table 3-2](#).

Return code ...	Indicates the ...
pmrcFNameTooLong	file name specified by the FileName parameter is too long.
pmrcBadFormat	value specified by the Format parameter is invalid.

## Usage Notes

The topics in the following table describe the things you should consider when specifying the file open parameters.

Topic	Usage Notes
File Open Function Requirements	A file open function is required when the Reqtype field of the pmiCmdBlock_t structure is pmiPIDMOpOpen.
Supporting Multiple Iterations	To maintain a connection between the file and the instance of the access module, use the PIData value with each file open request.

Topic	Usage Notes
The BlkSize Value	<p>The returned BlkSize value determines client buffering requirements. This value represents the maximum block size that will be sent to or received from the access module.</p> <p>On read operations, record lengths greater than the BlkSize specification would be considered an error condition that would overwrite client memory.</p> <p>On write operations, however, if the client utility needs to avoid spanned logical records, the access module should expect to typically receive blocks that are smaller than the BlkSize specification.</p>
The BlkHeaderLen Value	<p>The returned BlkHeaderLen value indicates the number of bytes at the beginning of each block returned on read operations that are access module specific.</p> <p>If, for example, the access module needs to prefix each record with a four-byte block length and that block length was present at the beginning of the returned read buffer, then the BlkHeaderLen would be set to 4.</p> <p>If, however, the access module does not include that field in its returned data, then the BlkHeaderLen value would be set to 0.</p>

# File Close

## Function

The file close function closes a file that is currently open for read or write operations.

**Definition of *file*:** The term *file* refers to a data source or destination on any sort of data storage device, such as magnetic tape, disk, CD-ROM, another database, and so forth, that the access module supports.

**Definition of *read* and *write*:** The terms *read* and *write* refer to data moving from (read) and to (write) the data source/destination.

## Structure

```

typedef struct _pmiClose
{
    char          EyeCatcher [pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32      StructLength;                /* Length of this structure */
    void          *FIData;
} pmiClose_t;

```

where,

Parameter ...	Is the ... field	That specifies the ...
EyeCatcher	input	structure description string, such as pmCloseParms.
StructLength	input	total structure length, including the EyeCatcher string.
FIData	input	value provided by the access module in response to a previous file open request call.

## Return Codes

The following file close return code supplements those listed in [Table 3-2](#).

Return code ...	Indicates ...
pmrcBadFp	invalid or meaningless value in FIData.

## Usage Notes

The topics in the following table describe the things you should consider when specifying the file close parameters.

Topic	Usage Notes
File Close Function Requirements	A file close function is required when the Reqtype field of the pmiCmdBlock_t structure is either: <ul style="list-style-type: none"><li data-bbox="776 499 1052 531">• pmiPIDMOptClose</li><li data-bbox="776 537 1068 569">• pmiPIDMOptCloseR</li></ul>
Removing/Releasing Media	If Reqtype is pmiPIDMOptCloseR, then the access module must also release/remove any associated removable media, such as dismounting a tape from a drive.

# File Read

## Function

The file read function reads a block of data from a currently open file.

## Structure

```
typedef struct _pmiRW
{
    char          EyeCatcher[pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32      StructLength;             /* Length of this structure */
    void          *FIData;
    pmUInt32      BufferLen;
    char          *Buffer;
} pmiRW_t;
```

where,

Parameter ...	Is the ... field	That specifies the ...
EyeCatcher	input	structure description string, such as pmReadParms.
StructLength	input	total structure length, including the EyeCatcher string.
FIData	input	value provided by the access module in response to a previous file open request call.
BufferLen	output	length of the returned data block.
Buffer	input	pointer to the buffer area into which the access module is to return the data block.

## Return Codes

The following file read return codes supplement those listed in [Table 3-2](#).

Return code ...	Indicates ...
pmrcBadFp	an invalid or meaningless value in FIData.
pmrcWriteOnly	the file specified by FIData is opened for write operations only.
pmrcEOF	an end-of-file was encountered. BufferLen is zero.

## Usage Notes

The topics in the following table describe the things you should consider when specifying the file read parameters.

Topic	Usage Notes
File Read Function Requirements	A file read function is required when the Reqtype field of the pmiCmdBlock_t structure is pmiPIDMOptRead.
The BlkHeaderLen Value	<p>The returned BlkHeaderLen value indicates the number of bytes at the beginning of each block returned on read operations that are access module specific.</p> <p>If, for example, the access module needs to prefix each record with a four-byte block length and that block length was present at the beginning of the returned read buffer, then the BlkHeaderLen would be set to 4.</p> <p>If, however, the access module does not include that field in its returned data, then the BlkHeaderLen value would be set to 0.</p>

# File Write

## Function

The file write function writes the provided block of data to a currently open file.

## Structure

```
typedef struct _pmiRW
{
    char        EyeCatcher[pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32    StructLength;              /* Length of this structure */
    void        *FIData;
    pmUInt32    BufferLen;
    char        *Buffer;
} pmiRW_t;
```

where,

Parameter ...	Is the ... field	That contains the ...
EyeCatcher	input	structure description string, such as pmWriteParms.
StructLength	input	total structure length, including the EyeCatcher string.
FIData	input	value provided by the access module in response to a previous open request call.
BufferLen	input	length of the data block to be written.
Buffer	input	pointer to the data block to be written.

## Return Codes

The following file write return codes supplement those listed in [Table 3-2](#).

Return code ...	Indicates ...
pmrcBadFp	an invalid or meaningless value in FIData.
pmrcReadOnly	the file indicated via FIData is opened for read operations only.

## Usage Notes

The topics in the following table describe the things you should consider when specifying the file write parameters.

Topic	Usage Notes
File Write Function Requirements	A file write function is required when the Reqtype field of the pmiCmdBlock_t structure is pmiPIDMOptWrite.

# File Get Position

## Function

The file get position function determines and returns the current media position of a currently open file.

## Structure

```
typedef struct _pmPosData
{
    pmUInt32      Length;
    char          *Data;
} pmPos_t;

typedef struct _pmiPos
{
    char          EyeCatcher[pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32      StructLength;             /* Length of this structure */
    void          *FIData;
    pmPos_t       Position;
} pmiPos_t;
```

where,

Parameter ...	Is the ... field	That specifies ...
EyeCatcher	input	the structure description string, such as pmGetParms.
StructLength	input	the total structure length, including the EyeCatcher string.
FIData	input	the value provided by the access module in response to a previous open request call.
Position	output	a two-field structure as follows: <ul style="list-style-type: none"> <li>Length—The length, in bytes, of the Data field</li> <li>Data—A pointer to the media position information</li> </ul>

## Return Codes

The file get position return codes are listed in [Table 3-2](#).

## Usage Notes

The topics in the following table describe the things you should consider when specifying the file get position parameters.

Topic	Usage Notes
File Get Position Function Requirements	A file get position function is required when the Rectype field of the pmiCmdBlock_t structure is pmiPIDMOptGetPos.
Position Data Length	Always minimize the length of the position data stored in the Length field of the pmPos_t structure, as certain arbitrary limits may be encountered.

# File Set Position

## Function

The file set position function repositions the media of a currently open file.

## Structure

```
typedef struct _pmPosData
{
    pmUInt32    Length;
    char        *Data;
} pmPos_t;

typedef struct _pmiPos
{
    char        EyeCatcher[pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32    StructLength;              /* Length of this structure */
    void        *FIData;
    pmPos_t     Position;
} pmiPos_t;
```

where,

Parameter ...	Is the ... field	That contains ...
EyeCatcher	input	the structure description string, such as pmSetParms.
StructLength	input	the total structure length, including the EyeCatcher string.
FIData	input	the value provided by the access module in response to a previous open request call.
Position	input	a two-field structure as follows: <ul style="list-style-type: none"> <li>Length—The length, in bytes, of the Data field</li> <li>Data—A pointer to the media position information</li> </ul>

## Return Codes

The file set position return codes are listed in [Table 3-2](#).

## Usage Notes

The topics in the following table describe the things you should consider when specifying the file set position parameters.

Topic	Usage Notes
File Set Position Function Requirements	A file set position function is required when the Rectype field of the pmiCmdBlock_t structure is pmiPIDMOptSetPos.

# Put Attribute

## Function

The put attribute function provides an arbitrary data value to the access module.

## Structure

```
typedef struct _pmiAttribute
{
    char        EyeCatcher [pmiMAX_EC_LEN];    /* Struct eyecatcher string */
    pmUInt32    StructLength;                  /* Length of this structure */
    void        *ObjData;                      /* Object pointer (PIData or FIData) */
    pmUInt32    AttrNameLen;                   /* Length of attribute name */
    char        AttrName [pmiMAX_ATR_NAME_LEN]; /* Attribute name */
    pmUInt32    AttrValueLen;                  /* Length of attribute value */
    char        AttrValue [pmiMAX_ATR_VAL_LEN]; /* Attribute value */
} pmiAttr_t;
```

where,

Parameter ...	Is the ... field	That contains the ...
EyeCatcher	input	structure description string, such as pmPutAParms.
StructLength	input	total structure length, including the EyeCatcher string.
ObjData	input	pointer to one of two forms of data.  The type of data depends on the value of Reqtype as follows: <ul style="list-style-type: none"> <li>pmiPIDMOptPutA_A—ObjData is a value provided by the access module in response to a previous access module initialization request call (as in PIData fields noted earlier)</li> <li>pmiPIDMOptPutF_A—ObjData is a value provided by the access module in response to a previous access module open request call (as in FIData fields noted earlier)</li> </ul>
AttrNameLen	input	length of the AttrName field.
AttrName	input	name of the attribute.
AttrValueLen	input	length of the AttrValue field.
AttrValue	input	value of the attribute.

## Return Codes

The following put attribute return code supplements those listed in [Table 3-2](#).

Return code . . .	Indicates . . .
pmrcBadAttrName	the AttrName is not recognized.

## Usage Notes

The topics in the following table describe the things you should consider when specifying the put attribute parameters.

Topic	Usage Notes
Put Attribute Function Requirements	A put attribute function is required when the Reqtype field of the pmiCmdBlock_t structure is pmiPIDMOptPutA_A or pmiPIDMOptPutF_A.
Client Utility to Access Module Communication	This function facilitates communication <i>from</i> the client utility <i>to</i> the access module. It might be used, for example, to send the name of the currently active database table to the access module.
Character Set Information	<p>Session character set is passed by pmAPutAttr() from the client utility to the access module for possible use. The attribute value is a variable-length character string with either the character set name or the character representation of the character set ID. The attribute varies based on how you specify the character set.</p> <p>If you specify the session character set by name, the attribute name is CHARSET_NAME.</p> <p>If you specify the session character set by ID, the attribute name is CHARSET_NUMBER.</p>

# Get Attribute

## Function

The get attribute function retrieves an arbitrary data value from the access module.

## Structure

```
typedef struct _pmiAttribute
{
    char      EyeCatcher [pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32  StructLength;                /* Length of this structure */
    void      *ObjData;                    /* Object pointer (PIData or FIData) */
    pmUInt32  AttrNameLen;                 /* Length of attribute name */
    char      AttrName [pmiMAX_ATR_NAME_LEN]; /* Attribute name */
    pmUInt32  AttrValueLen;                /* Length of attribute value */
    char      AttrValue [pmiMAX_ATR_VAL_LEN]; /* Attribute value */
} pmiAttr_t;
```

where,

Parameter ...	Is the ... field	That contains the ...
EyeCatcher	input	structure description string, such as pmGetAParms.
StructLength	input	total structure length, including the EyeCatcher string.
ObjData	input	pointer to one of two forms of data.  The type of data depends on the value of Reqtype as follows: <ul style="list-style-type: none"> <li>pmiPIDMOptGetA_A—ObjData is a value provided by the access module in response to a previous initialization request call (as in PIData fields noted earlier)</li> <li>pmiPIDMOptGetF_A—ObjData is a value provided by the access module in response to a previous open request call (as in FIData fields noted earlier)</li> </ul>
AttrNameLen	input	length of the AttrName field.
AttrName	input	name of the attribute.
AttrValueLen	output	length of the AttrValue field.
AttrValue	output	value of the attribute.

## Return Codes

The following get attribute return code supplements those listed in [Table 3-2](#).

Return code . . .	Indicates . . .
pmrcBadAttrName	the AttrName is not recognized.

## Usage Notes

The topics in the following table describe the things you should consider when specifying the get attribute parameters.

Topic	Usage Notes
Get Attribute Function Requirements	A get attribute function is required when the Reqtype field of the pmiCmdBlock_t structure is pmiPIDMOptGetA_A or pmiPIDMOptGetF_A.
Access Module to Client Utility Communication	This function facilitates communication <i>to</i> the client utility <i>from</i> the access module. It might be used, for example, to send a physical volume identification to the client utility.

# Shutdown

## Function

The shutdown function terminates the access module.

## Structure

```

typedef struct _pmiShut
{
    char          EyeCatcher[pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32      StructLength;              /* Length of this structure */
    void          *PIData;
    pmUInt16      Mode;
} pmiShut_t;

```

where,

Parameter ...	Is the ... field	That contains the ...
EyeCatcher	input	structure description string, such as pmShutParms.
StructLength	input	total structure length, including the EyeCatcher string.
PIData	input	value provided by the access module in response to a previous open request call.
Mode	input	shutdown mode/option as follows: <ul style="list-style-type: none"> <li>pmShutDownNormal—Normal error checking. If anything prevents a <i>clean</i> termination (such as files still open), then exit without doing anything with a non-pmrcOK in Retcode.</li> <li>pmShutDownBrute—Take arbitrary action for any unexpected conditions.</li> </ul>

## Return Codes

The following shutdown return code supplements those listed in [Table 3-2](#):

Return code ...	Indicates ...
pmrcOpenFiles	there are files open.

**Shutdown****Usage Notes**

The topics in the following table describe the things you should consider when specifying the shutdown parameters.

Topic	Usage Notes
Shutdown Function Requirements	A shutdown function is required when the Rectype field of the pmiCmdBlock_t structure is pmiPIDMOptShut.
Housekeeping Operations	The shutdown function should also perform any housekeeping operations, such as: <ul data-bbox="776 619 1177 730" style="list-style-type: none"><li>• Freeing memory</li><li>• Closing currently open files</li><li>• Terminating any subprocesses</li></ul>
Error Conditions	The shutdown function should proceed despite any file close errors.



**Appendix A:**

## Access Module Header File Examples

This appendix provides four header file examples:

- `pmdcomt.h` on [page A-2](#), for access modules
- `pmddamt.h` on [page A-6](#), for Data Connector Application Program Interface (API) modules
- `pmdcomt.h` on [page A-11](#), for use with the Teradata Warehouse Builder DataConnector operator
- `pmddamt.h` on [page A-14](#), for use with the Teradata Warehouse Builder DataConnector operator module

---

# Access Modules

## pmdcomt.h

This header file shows example client-related structure and symbol definitions relating to access modules.

```
/* % TITLE pmdcomt.h ... Independent/Dependent Access Module */
/*                               common struct/types */
/*****
/*                               */
/* COPYRIGHT 1997-2002, NCR Corporation. ALL RIGHTS RESERVED. */
/* This copyrighted material is the confidential, unpublished property of */
/* NCR Corp. This copyright notice and any other copyright notices */
/* included in machine readable copies must be reproduced on all authorized */
/* copies. */
/*                               */
/*****

#ifndef PMDCOMT_H
#define PMDCOMT_H

#define pmMAX_CLIENT_ID      255
#define pmMAX_ATR_NAME_LEN  255
#define pmMAX_ATR_VAL_LEN   255

#if defined(HPUX) && defined(PIOM64)
#define pmdcomt_packing "on"
#ifdef __cplusplus
#pragma pack 8
#else
#pragma HP_ALIGN NATURAL PUSH
#endif
#elif defined(SOLARIS) && defined(PIOM64)
#define pmdcomt_packing "on"
#pragma pack(8)
#elif !defined(AIX) && !defined(HPUX) && !defined(SOLARIS) \
    && !defined(_WIN64)
#define pmdcomt_packing "on"
#pragma pack (push, 1)
#endif

#if !defined (pmdcomt_packing)
#define pmdcomt_packing "off"
#endif

#define pmdcomt_HeaderVersion "Standalone 02.00.00.05"
```

```

#ifdef PIOM64
#define pmInterfaceVersionD 1001
typedef unsigned intpmUInt32;      /* 32-bit unsigned integer */
#else
#define pmInterfaceVersionD 1000
typedef unsigned longpmUInt32;     /* 32-bit unsigned integer */
#endif

typedef unsigned short pmUInt16;    /* 16-bit unsigned integer */
typedef pmUInt32 pmReturnType;     /* Return value type for */
/* all API functions */
typedef shortpmOpenMode_t; /* Access Module mode for pmOpen */
typedef int pmTrceLvl_t; /* Trace level type */

/* Media Position description structure */
typedef struct _pmPosData
{
    pmUInt32Length;
    char*Data;
} pmPos_t;

typedef struct _pmNameBuf
{
    pmUInt32DataLength;
    char*Data;
} pmNameBuf_t;

/* Access Module Version Identification structure */
#define pmMAX_VER_STR_LEN 31
typedef struct _pmVerLst_t
{
    char ModuleName [pmMAX_VER_STR_LEN+1];
    char ModuleVers [pmMAX_VER_STR_LEN+1];
    struct _pmVerLst_t *Next;
} pmVerLst_t;

/* List of return codes generated by any call to the API */
#define pmrcOK 0 /* All's well :: MUST BE ZERO !! :: */
#define pmrcBadpmHandle 1 /* Invalid pmHandle parameter passed */
#define pmrcBadParm 2 /* Bad parameter passed to API */
#define pmrcAXMNotFound 3 /* Requested Access Module not found */
#define pmrcFileNotFound 4 /* Requested file not found */
#define pmrcWriteOnly 5 /* Access Module is write only */
#define pmrcReadOnly 6 /* Access Module is read only */
#define pmrcBadFp 7 /* Invalid pmFp parameter passed */
#define pmrcOpenFiles 8 /* Access Module still has open files */
#define pmrcEOF 9 /* Access Module reached EOF */
#define pmrcCPReady 10 /* Access Module is ready to checkpoint */

```

## Appendix A: Access Module Header File Examples

### Access Modules

```
#define pmrcInvUtil      11  /* Invalid utility ID passed to pmInit */
#define pmrcAllocErr    12  /* Error encountered during memory mgmt */
#define pmrcBadOpenMode 13  /* Unsupported mode */
#define pmrcBadFormat   14  /* Unsupported format */
#define pmrcBadBlksize  15  /* Unsupported block size */
#define pmrcDataFormatErr 16 /* Unexpected data format */
#define pmrcBufferOverflow 17 /* Record beyond buffer */
#define pmrcBadVer      18  /* info->InterfaceVerNo!=pmInterfaceVersion */
#define pmrcFNameTooLong 19 /* File name too long */
#define pmrcInitCmdSyntax 20 /* Access Module Init string syntax error */
#define pmrcInitInvCmd    21 /* Init string invalid command char */
#define pmrcInitInvOpt    22 /* Init string invalid cmd option */
#define pmrcPosDataInvL   23 /* Invalid positioning data length */
#define pmrcUnsupported   24 /* Unsupported feature */
#define pmrcKeepReading   25 /* Continue reading 'till end-of-buffer */
#define pmrcAXMnotReady   26 /* pmInit has not been called */
#define pmrcBadTraceLvl  27 /* Invalid diagnostic trace level */
#define pmrcOpenAXMs     28 /* Access Modules still open */
#define pmrcEmptyFile    29 /* Empty file on read open */
#define pmrcRedundant     30 /* Redundant pmInit call */
#define pmrcBadFormatInfo 31 /* Invalid Supplemental Format Information */
#define pmrcBadChkPtMode  32 /* Invalid Checkpointing mode */
#define pmrcBadAttrName   33 /* Unrecognized attribute name */
#define pmrcFailure       34 /* Indeterminate error.  Ref pmGetErrText() */
#define pmrcNoEOR        35 /* EOF before EOR on text data */

/* Open modes (Read, Write, or ReadWrite) */
#define pmOpenModeR  1 /* Read only */
#define pmOpenModeW  2 /* Write only */
#define pmOpenModeRW 3 /* Read first, then write only */
#define pmOpenMode_MAX 3

/* Shutdown modes */
#define pmShutDownNormal 1
#define pmShutDownBrute  2

/* Trace levels */
#define pmTrceNone 1 /* issue no trace messages */
#define pmTrceEvents 2
#define pmTrceIOcounts 3
#define pmTrceIObufs 4
#define pmTrceInfo 5
#define pmTrce_MAX 5

#if defined(HPUX) && defined(PIOM64)
#ifndef __cplusplus
#pragma pack
#else
#pragma HP_ALIGN POP
#endif
#endif
```

```
#endif
#elif defined(SOLARIS) && defined(PIOM64)
#pragma pack()
#elif !defined(AIX) && !defined(HPUX) && !defined(SOLARIS) \
      && !defined(_WIN64)
#pragma pack (pop)
#endif

#endif /* PMDCOMT_H */
```

---

# Data Connector APIs

## pmddamt.h

This header file shows structure and symbol definitions required to interface with the Data Connector API modules.

```
/* % TITLE pmddamt.h ... Device Dependent Access module structs/types      */
/*****                                                                    */
/*                                                                            */
/* COPYRIGHT 1997-2002, NCR Corporation.  ALL RIGHTS RESERVED.             */
/* This copyrighted material is the confidential, unpublished property of   */
/* NCR Corp.  This copyright notice and any other copyright notices        */
/* included in machine readable copies must be reproduced on all authorized */
/* copies.                                                                    */
/*                                                                            */
/*****                                                                    */

#ifndef PMDDAMT_H
#define PMDDAMT_H

#if defined(HPUX) && defined(PIOM64)
#define pmddamt_packing "on"
#ifdef __cplusplus
#pragma pack 8
#else
#pragma HP_ALIGN NATURAL PUSH
#endif
#elif defined(SOLARIS) && defined(PIOM64)
#define pmddamt_packing "on"
#pragma pack(8)
#elif !defined(AIX) && !defined(HPUX) && !defined(SOLARIS) \
      && !defined(_WIN64)
#define pmddamt_packing "on"
#pragma pack (push, 1)
#endif

#if !defined (pmddamt_packing)
#define pmddamt_packing "off"
#endif

#define pmddamt_HeaderVersion "Standalone 02.00.00.05"

#ifdef PIOM64
#define pmiInterfaceVersion 2001
#else
```

```

#define pmiInterfaceVersion    2000
#endif

#define pmiMAX_EC_LEN        30
/*                               EyeCatcher strings for each structure type */
#define pmiEC_CmdBlock_t     "AXSMOD block:CmdBlock_t"
#define pmiEC_Init_t         "AXSMOD block:Init_t"
#define pmiEC_Open_t         "AXSMOD block:Open_t"
#define pmiEC_Close_t        "AXSMOD block:Close_t"
#define pmiEC_RW_t           "AXSMOD block:RW_t"
#define pmiEC_Pos_t          "AXSMOD block:Pos_t"
#define pmiEC_Shut_t         "AXSMOD block:Shut_t"
#define pmiEC_ID_t           "AXSMOD block:ID_t"
#define pmiEC_Attr_t         "AXSMOD block:Attr_t"

#define pmiMAX_INIT_STR_LEN  512
#define pmiMAX_FNAME_LEN     255
#define pmiMAX_ETEXT_LEN     32000

/*                               the following defines are required only for */
/*                               communications between the DIAM and the DDAM */
#define pmiPIDMOptInit        1
#define pmiPIDMOptOpen        2
#define pmiPIDMOptClose       3
#define pmiPIDMOptCloseR      10
#define pmiPIDMOptRead        4
#define pmiPIDMOptWrite       5
#define pmiPIDMOptGetPos      6
#define pmiPIDMOptSetPos      7
#define pmiPIDMOptShut        8
#define pmiPIDMOptID          9
#define pmiPIDMOptGetA_A      11
#define pmiPIDMOptGetF_A      12
#define pmiPIDMOptPutA_A      13
#define pmiPIDMOptPutF_A      14

typedef struct _pmiInit
{
    char        EyeCatcher[pmiMAX_EC_LEN];    /* Struct eyecatcher string */
    pmUInt32    StructLength;                 /* Length of this structure */
    void        *PIData;                      /* Access module Internal Data Pointer */
    pmUInt16    InterfaceVerNo;               /* pmdcomt.h version identifier */
    pmUInt16    InterfaceVerNoD;              /* pmddamt.h version identifier */
    pmUInt32    ClientIDL;                    /* Length of ClientID field */
    pmUInt32    InitStrL;                     /* Length of InitStr field */
    char        ClientID[pmMAX_CLIENT_ID+1]; /* Client identifier */
    char        InitStr[pmiMAX_INIT_STR_LEN+1]; /* Initialization field */
} pmiInit_t;

```

**Data Connector APIs**

```
typedef struct _pmiOpen
{
    char            EyeCatcher [pmiMAX_EC_LEN];          /* Struct eyecatcher string */
    pmUInt32       StructLength;                        /* Length of this structure */
    void           *PIData;                             /* Access module Internal Data Pointer */
    pmOpenMode_t   OpenMode;                            /* provided to PIDM */
    pmUInt16       BlkHeaderLen;                        /* defined by PIDM */
    pmUInt32       BlkSize;                             /* defined by PIDM */
    void           *FIData;                             /* File Internal Data (returned/used by PIDM) */
    pmUInt16       FileNameL;                           /* provided to PIDM */
    char           FileName [pmiMAX_FNAME_LEN+1];      /* provided to PIDM */
} pmiOpen_t;
```

```
typedef struct _pmiClose
{
    char            EyeCatcher [pmiMAX_EC_LEN];          /* Struct eyecatcher string */
    pmUInt32       StructLength;                        /* Length of this structure */
    void           *FIData;
} pmiClose_t;
```

```
typedef struct _pmiRW
{
    char            EyeCatcher [pmiMAX_EC_LEN];          /* Struct eyecatcher string */
    pmUInt32       StructLength;                        /* Length of this structure */
    void           *FIData;
    pmUInt32       BufferLen;
    char           *Buffer;
} pmiRW_t;
```

```
typedef struct _pmiPos
{
    char            EyeCatcher [pmiMAX_EC_LEN];          /* Struct eyecatcher string */
    pmUInt32       StructLength;                        /* Length of this structure */
    void           *FIData;
    pmPos_t        Position;
} pmiPos_t;
```

```
typedef struct _pmiShut
{
    char            EyeCatcher [pmiMAX_EC_LEN];          /* Struct eyecatcher string */
    pmUInt32       StructLength;                        /* Length of this structure */
    void           *PIData;
    pmUInt16       Mode;
} pmiShut_t;
```

```
typedef struct _pmiID
{
    char            EyeCatcher [pmiMAX_EC_LEN];          /* Struct eyecatcher string */
    pmUInt32       StructLength;                        /* Length of this structure */
}
```

```

    pmVerLst_t VerIDList;
} pmiID_t;

typedef struct _pmiAttribute
{
    char        EyeCatcher[pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32    StructLength;              /* Length of this structure */
    void        *ObjData;                  /* Object pointer (PIData or FIData) */
    pmUInt32    AttrNameLen;               /* Length of attribute name */
    char        AttrName[pmMAX_ATR_NAME_LEN]; /* Attribute name */
    pmUInt32    AttrValueLen;              /* Length of attribute value */
    char        AttrValue[pmMAX_ATR_VAL_LEN]; /* Attribute value */
} pmiAttr_t;

typedef struct _pmiCmdBlock
{
    char        EyeCatcher[pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32    StructLength;              /* Length of this structure */
    pmUInt32    Reqtype;
    pmReturnType Retcode;                  /* set by DDAM */
    pmTrceLvl_t TraceLvl;
    pmNameBuf_t ErrMsg;                    /* Set by DDAM if retcode is pmrcFailure */
} pmiCmdBlock_t;

#ifdef WIN32
#ifdef CPP
extern "C"
    {
        __declspec(dllexport) void PIDMMain ( pmiCmdBlock_t *, void * );
    }
#else
__declspec(dllexport) void PIDMMain ( pmiCmdBlock_t *, void * );
#endif
#endif

void PIDMMain ( pmiCmdBlock_t *, void * );
#endif

#ifdef HPUX && defined(PIOM64)
#ifdef __cplusplus
#pragma pack
#else
#pragma HP_ALIGN POP
#endif
#elif defined(SOLARIS) && defined(PIOM64)
#pragma pack()
#elif !defined(AIX) && !defined(HPUX) && !defined(SOLARIS) \
    && !defined(_WIN64)

```

Appendix A: Access Module Header File Examples  
**Data Connector APIs**

```
#pragma pack (pop)  
#endif  
  
#endif /* PMDDAMT_H */
```

# Teradata Warehouse Builder DataConnector Operator

## pmdcomt.h

This header file shows example client-related structure and symbol definitions relating to access modules.

```

/* % TITLE pmdcomt.h ... Independent/Dependent Access Module          */
/*                               common struct/types                    */
/*****                                                                    */
/*                               */
/* COPYRIGHT 1998-2002, NCR Corporation.  ALL RIGHTS RESERVED.        */
/* This copyrighted material is the confidential, unpublished property of */
/* NCR Corp.  This copyright notice and any other copyright notices     */
/* included in machine readable copies must be reproduced on all authorized */
/* copies.                                                                */
/*                               */
/* For use with the Teradata Warehouse Builder DataConnector operator.  */
/*                               */
/*****                                                                    */

#ifndef PMDCOMT_H
#define PMDCOMT_H

#ifdef USING_PACK
#define pmdcomt_packing "on"
#if ( (defined __linux__) || (defined __MVS__) )
#pragma pack(1)
#else
#pragma pack (push, 1)
#endif
#endif

#if !defined (pmdcomt_packing)/
#define pmdcomt_packing "off"
#endif

#define pmdcomt_HeaderVersion "TWB 03.00.00.02"
#define pmInterfaceVersionD 1000

#define pmMAX_CLIENT_ID      255
#define pmMAX_ATR_NAME_LEN  255
#define pmMAX_ATR_VAL_LEN   255

typedef unsigned longpmUInt32;      /* 32-bit unsigned integer */
typedef unsigned short   pmUInt16;  /* 16-bit unsigned integer */

```

Appendix A: Access Module Header File Examples  
**Teradata Warehouse Builder DataConnector Operator**

```
typedef pmUInt32      pmReturnType; /* Return value type for */
                        /* all API functions */
typedef short        pmOpenMode_t; /* Access Module mode for pmOpen */
typedef int          pmTrceLvl_t; /* Trace level type */

/* Media Position description structure */
typedef struct _pmPosData
{
    pmUInt32Length;
    char*Data;
} pmPos_t;

typedef struct _pmNameBuf
{
    pmUInt32DataLength;
    char*Data;
} pmNameBuf_t;

/* Access Module Version Identification structure */
#define pmMAX_VER_STR_LEN 31
typedef struct _pmVerLst_t
{
    char          ModuleName [pmMAX_VER_STR_LEN+1];
    char          ModuleVers [pmMAX_VER_STR_LEN+1];
    struct _pmVerLst_t *Next;
} pmVerLst_t;

/* List of return codes generated by any call to the API */
#define pmrcOK                0 /* All's well :: MUST BE ZERO !! :: */
#define pmrcBadpmHandle      1 /* Invalid pmHandle parameter passed */
#define pmrcBadParm          2 /* Bad parameter passed to API */
#define pmrcAXMNotFound       3 /* Requested Access Module not found */
#define pmrcFileNotFound     4 /* Requested file not found */
#define pmrcWriteOnly        5 /* Access Module is write only */
#define pmrcReadOnly         6 /* Access Module is read only */
#define pmrcBadFp            7 /* Invalid pmFp parameter passed */
#define pmrcOpenFiles        8 /* Access Module still has open files */
#define pmrcEOF              9 /* Access Module reached EOF */
#define pmrcCPReady         10 /* Access Module is ready to checkpoint */
#define pmrcInvUtil          11 /* Invalid utility ID passed to pmInit */
#define pmrcAllocErr         12 /* Error encountered during memory mgmt */
#define pmrcBadOpenMode     13 /* Unsupported mode */
#define pmrcBadFormat        14 /* Unsupported format */
#define pmrcBadBlksize       15 /* Unsupported block size */
#define pmrcDataFormatErr    16 /* Unexpected data format */
#define pmrcBufferOverFlow   17 /* Record beyond buffer */
#define pmrcBadVer           18 /* info->InterfaceVerNo!=pmInterfaceVersion */
#define pmrcFNameTooLong    19 /* File name too long */
#define pmrcInitCmdSyntax    20 /* Access Module Init string syntax error */
```

```

#define pmrcInitInvCmd      21  /* Init string invalid command char */
#define pmrcInitInvOpt     22  /* Init string invalid cmd option */
#define pmrcPosDataInvL    23  /* Invalid positioning data length */
#define pmrcUnsupported    24  /* Unsupported feature */
#define pmrcKeepReading    25  /* Continue reading 'till end-of-buffer */
#define pmrcAXMnotReady    26  /* pmInit has not been called */
#define pmrcBadTraceLvl    27  /* Invalid diagnostic trace level */
#define pmrcOpenAXMs      28  /* Access Modules still open */
#define pmrcEmptyFile     29  /* Empty file on read open */
#define pmrcRedundant      30  /* Redundant pmInit call */
#define pmrcBadFormatInfo  31  /* Invalid Supplemental Format Information */
#define pmrcBadChkPtMode   32  /* Invalid Checkpointing mode */
#define pmrcBadAttrName    33  /* Unrecognized attribute name */
#define pmrcFailure        34  /* Indeterminate error. Ref pmGetErrText() */
#define pmrcNoEOR          35  /* No EOR on last text record */
#define pmrcBufferSizeExcess 36 /* BufferSize exceeded by axsmode */

/* Open modes (Read, Write, or ReadWrite) */
#define pmOpenModeR      1      /* Read only */
#define pmOpenModeW      2      /* Write only */
#define pmOpenModeRW     3      /* Read first, then write only */
#define pmOpenMode_MAX  3

/* Shutdown modes */
#define pmShutDownNormal  1
#define pmShutDownBrute   2

/* Trace levels */
#define pmTrceNone        1 /* issue no trace messages */
#define pmTrceEvents      2
#define pmTrceIOcounts    3
#define pmTrceIObufs      4
#define pmTrceInfo        5
#define pmTrce_MAX        5

#ifdef USING_PACK
#if ( ! ( (defined __linux__) || (defined __MVS__) ) )
#pragma pack (pop)
#endif
#endif

#endif /* PMDCOMT_H */

```

---

# Teradata Warehouse Builder DataConnector Operator Module

## pmddamt.h

This header file shows structure and symbol definitions required to interface with the TWB Data Connector Operator module.

```
/* % TITLE pmddamt.h ... Device Dependent Access module structs/types */
/*****
/*
/* COPYRIGHT 1998-2002, NCR Corporation. ALL RIGHTS RESERVED. */
/* This copyrighted material is the confidential, unpublished property of */
/* NCR Corp. This copyright notice and any other copyright notices */
/* included in machine readable copies must be reproduced on all authorized */
/* copies. */
/*
/* For use with the Teradata Warehouse Builder DataConnector operator. */
/*
*****/

#ifndef PMDDAMT_H
#define PMDDAMT_H

#ifdef USING_PACK
#define pmddamt_packing "on"
#if ( (defined __linux__) || (defined __MVS__) )
#pragma pack(1)
#else
#pragma pack (push, 1)
#endif
#endif

#if !defined (pmddamt_packing)
#define pmddamt_packing "off"
#endif

#define pmddamt_HeaderVersion "TWB 03.00.00.02"
#define pmiInterfaceVersion 2000

#define pmiMAX_EC_LEN 30
/* EyeCatcher strings for each structure type */
#define pmiEC_CmdBlock_t "AXSMOD block:CmdBlock_t"
#define pmiEC_Init_t "AXSMOD block:Init_t"
#define pmiEC_Open_t "AXSMOD block:Open_t"
#define pmiEC_Close_t "AXSMOD block:Close_t"
#define pmiEC_RW_t "AXSMOD block:RW_t"
```

```

#define pmiEC_Pos_t           "AXSMOD block:Pos_t"
#define pmiEC_Shut_t         "AXSMOD block:Shut_t"
#define pmiEC_ID_t           "AXSMOD block:ID_t"
#define pmiEC_Attr_t         "AXSMOD block:Attr_t"

#define pmiMAX_INIT_STR_LEN  512
#define pmiMAX_FNAME_LEN     255
#define pmiMAX_ETEXT_LEN     32000

/*                               the following defines are required only for */
/*                               communications between the DIAM and the DDAM */
#define pmiPIDMOptInit       1
#define pmiPIDMOptOpen       2
#define pmiPIDMOptClose      3
#define pmiPIDMOptCloseR    10
#define pmiPIDMOptRead       4
#define pmiPIDMOptWrite      5
#define pmiPIDMOptGetPos     6
#define pmiPIDMOptSetPos    7
#define pmiPIDMOptShut       8
#define pmiPIDMOptID         9
#define pmiPIDMOptGetA_A    11
#define pmiPIDMOptGetF_A    12
#define pmiPIDMOptPutA_A    13
#define pmiPIDMOptPutF_A    14

typedef struct _pmiInit
{
    char      EyeCatcher[pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32  StructLength;              /* Length of this structure */
    void      *PIData;                   /* Access module Internal Data Pointer */
    pmUInt16  InterfaceVerNo;            /* pmdcomt.h version identifier */
    pmUInt16  InterfaceVerNoD;           /* pmddamt.h version identifier */
    pmUInt32  ClientIDL;                 /* Length of ClientID field */
    pmUInt32  InitStrL;                  /* Length of InitStr field */
    char      ClientID[pmMAX_CLIENT_ID+1]; /* Client identifier */
    char      InitStr[pmiMAX_INIT_STR_LEN+1]; /* Initialization field */
} pmiInit_t;

typedef struct _pmiOpen
{
    char      EyeCatcher[pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32  StructLength;              /* Length of this structure */
    void      *PIData;                   /* Access module Internal Data Pointer */
    pmOpenMode_t OpenMode;                /* provided to PIDM */
    pmUInt16  BlkHeaderLen;               /* defined by PIDM */
    pmUInt32  BlkSize;                   /* defined by PIDM */
    void      *FIData;                   /* File Internal Data (returned/used by PIDM) */
    pmUInt16  FileNameL;                 /* provided to PIDM */
}

```

**Teradata Warehouse Builder DataConnector Operator Module**

```
char      FileName [pmiMAX_FNAME_LEN+1];          /* provided to PIDM */
} pmiOpen_t;
```

```

typedef struct _pmiClose
{
    char        EyeCatcher [pmiMAX_EC_LEN];    /* Struct eyecatcher string */
    pmUInt32    StructLength;                  /* Length of this structure */
    void        *FIData;
} pmiClose_t;

typedef struct _pmiRW
{
    char        EyeCatcher [pmiMAX_EC_LEN];    /* Struct eyecatcher string */
    pmUInt32    StructLength;                  /* Length of this structure */
    void        *FIData;
    pmUInt32    BufferLen;
    char        *Buffer;
} pmiRW_t;

typedef struct _pmiPos
{
    char        EyeCatcher [pmiMAX_EC_LEN];    /* Struct eyecatcher string */
    pmUInt32    StructLength;                  /* Length of this structure */
    void        *FIData;
    pmPos_t     Position;
} pmiPos_t;

typedef struct _pmiShut
{
    char        EyeCatcher [pmiMAX_EC_LEN];    /* Struct eyecatcher string */
    pmUInt32    StructLength;                  /* Length of this structure */
    void        *PIData;
    pmUInt16    Mode;
} pmiShut_t;

typedef struct _pmiID
{
    char        EyeCatcher [pmiMAX_EC_LEN];    /* Struct eyecatcher string */
    pmUInt32    StructLength;                  /* Length of this structure */
    pmVerLst_t VerIDList;
} pmiID_t;

typedef struct _pmiAttribute
{
    char        EyeCatcher [pmiMAX_EC_LEN];    /* Struct eyecatcher string */
    pmUInt32    StructLength;                  /* Length of this structure */
    void        *ObjData;                      /* Object pointer (PIData or FIData) */
    pmUInt32    AttrNameLen;                   /* Length of attribute name */
    char        AttrName [pmMAX_ATR_NAME_LEN]; /* Attribute name */
    pmUInt32    AttrValueLen;                  /* Length of attribute value */
    char        AttrValue [pmMAX_ATR_VAL_LEN]; /* Attribute value */
} pmiAttr_t;

```

Appendix A: Access Module Header File Examples  
**Teradata Warehouse Builder DataConnector Operator Module**

```
typedef struct _pmiCmdBlock
{
    char            EyeCatcher[pmiMAX_EC_LEN]; /* Struct eyecatcher string */
    pmUInt32        StructLength;             /* Length of this structure */
    pmUInt32        Reqtype;
    pmReturnType    Retcode;                 /* set by DDAM */
    pmTrceLvl_t     TraceLvl;
    pmNameBuf_t     ErrMsg;                  /* Set by DDAM if retcode is pmrcFailure */
} pmiCmdBlock_t;

#ifdef WIN32
#ifdef CPP
extern "C"
    {
        __declspec(dllexport) void PIDMMain ( pmiCmdBlock_t *, void * );
    }
#else
__declspec(dllexport) void PIDMMain ( pmiCmdBlock_t *, void * );
#endif
#else
#ifdef __cplusplus
extern "C"
#endif
void PIDMMain ( pmiCmdBlock_t *, void * );
#endif

#ifdef USING_PACK
#if (!( (defined __linux__) || (defined __MVS__) ))
#pragma pack (pop)
#endif
#endif

#endif /* PMDDAMT_H */
```



**Appendix B:**

## Access Module Examples

This appendix provides listings of a sample access module program and makefile.

---

# Sample Program

## EGaxsm.c.

```
/* % TITLE EGaxsm.c ... Sample Access Module
*/
/*****
*/
/*          Copyright 1998, by NCR Corporation.  All Rights Reserved.          */
/*          NCR PROPRIETARY AND CONFIDENTIAL--RESTRICTED                        */
/*          */                                                                    */
/* Purpose:  Provide an Access Module template                                */
/*          */                                                                    */
/* Contents:                                                                 */
/*          PIDMMain - receives all Access Module calls from the NCR          */
/*                  Teradata Client Utility via the Data Connector.            */
/*                  It provides rudimentary disk I/O support.                  */
/*          */                                                                    */
/* History Information                                                         */
/*          */                                                                    */
/*          DR/#                                                                 */
/* Revision   Date      DCR   DID   Comments                                */
/* -----   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 01.01.00.01 08071998 ----- TWB   Terminology changes.                */
/* 01.00.00.01 05291998 ----- TWB   Initial version                      */
/*          */                                                                    */
/*****
*/

#include <stdio.h>
#ifdef WIN32
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <windows.h>
#endif
#include "pmdcomt.h"
#include "pmdamt.h"

#define MODULE_NAME "EGAXSM"
#define MODULE_VERS "01.01.00.01"

#define AXSM_DEFAULT_BLKSIZE 32000

static char          SG_ErrorText [pmiMAX_ETEXT_LEN+1];
static size_t       SG_ErrorTextLen=0;
```

```

static char      AXMAttr_1[]  = "AXSM AXM attribute #1 value";
static char      FileAttr_1[] = "AXSM File attribute #1 value";

/*****
/*****
/*
/*                               Main routine */
/*****
/*****
#ifdef WIN32
__declspec(dllexport)
#endif
void PIDMMain ( pmiCmdBlock_t *Opts, void *OptParms )
{
    pmReturnType      retcode=pmrcOK;
    char              *cptr;
    static FILE       *Fp = 0;
    char*             oOption = NULL;
    static long       FileOffset, TmpOffset;
    size_t            frc;
    unsigned long     amtWritten = 0;
    extern int        errno; /* system error indicator */
    int               seekrc;
    char              AXSM_FileName[pmiMAX_FNAME_LEN+1];

/*****
/*
/*                               This is the main decision block */
/*
/*                               Specific action taken is determined by the value */
/*
/*                               value of the Reqtype field in the Opts parameter */
/*****

    switch (Opts->Reqtype) /* switch on request type */
    {
/*****-----*/
/*
/*                               AXSM Initialization */
/*****-----*/
        case pmiPIDMOptInit:

/*****
/*                               Make sure the headers used by the calling DIAM */
/*
/*                               are the same versions the this AXSM used */
/*
/*                               when compiling! */
/*
/*                               Otherwise, NOTHING CAN BE TRUSTED!! */

            if ( ((pmiInit_t*)OptParms)->InterfaceVerNo != pmInterfaceVersionD
                || ((pmiInit_t*)OptParms)->InterfaceVerNoD != pmInterfaceVersion )
                retcode = pmrcBadVer;

/*****
/*                               Set that all important AXSM identifying void pointer */
/*
/*                               that the AXSM can use for anything */

```

Appendix B: Access Module Examples  
Sample Program

```
        ((pmiInit_t*)OptParms)->PIData = 0;
    break;
/*-----*/
/*                                     File Open */
/*-----*/
    case pmiPIDMOptOpen:

/*                                     Make sure that NO file has been opened */
    if ( Fp )
    {
        strcpy (SG_ErrorText,"DUMMY Access Module limited to 1 open file");
        SG_ErrorTextLen = strlen(SG_ErrorText);
        retcode = pmrcFailure;
    }
    if ( retcode ) break;

/*                                     Make sure that the requested open mode is supported by the AXSM */

    switch ( ((pmiOpen_t*)OptParms)->OpenMode)
    {
        case pmOpenModeR:
            oOption = "rb";
            break;
        case pmOpenModeW:
            oOption = "wb";
            break;
        case pmOpenModeRW:
            oOption = "rb+";
            break;
        default:
            retcode = pmrcUnsupported;
    }
    if ( retcode ) break;

/*                                     Watch that file name length! */
/*                                     strcpys can be dangerous! */

    if ( ((pmiOpen_t*)OptParms)->FileNameL > pmiMAX_FNAME_LEN )
    {
        retcode = pmrcFNameTooLong;
    }
    if ( retcode ) break;

/*                                     Remember, there's no guarantee that the name is NULL terminated */
/*                                     If you expect it so, do it here */
    strncpy ( AXSM_FileName,
              ((pmiOpen_t*)OptParms)->FileName,
              ((pmiOpen_t*)OptParms)->FileNameL);
```

```

AXSM_FileName[((pmiOpen_t*)OptParms)->FileNameL] = '\0';

/*                                     Do the open operation */
/*                                     and be sure to check for errors */

if ((Fp = fopen(AXSM_FileName,oOption)) == NULL)
{
    strcpy (SG_ErrorText,"Unknown Open failure");
    SG_ErrorTextLen = strlen(SG_ErrorText);
    retcode = pmrcFailure;
}

/*                                     Return device max block size, internal header length */
/*                                     and that all important file identifying void pointer */
/*                                     that the Access Module can use for anything */

((pmiOpen_t*)OptParms)->BlkSize = AXSM_DEFAULT_BLKSIZE;
((pmiOpen_t*)OptParms)->BlkHeaderLen = 0;
((pmiOpen_t*)OptParms)->FIData = 0;
break;
/*-----*/
/*                                     File Close */
/*-----*/
case pmiPIDMOptClose:
case pmiPIDMOptCloser:

/*                                     Make sure that the file has been opened */
if ( !Fp )
{
    strcpy (SG_ErrorText,"Attempt to close, no File Open");
    SG_ErrorTextLen = strlen(SG_ErrorText);
    retcode = pmrcFailure;
}
if ( retcode ) break;

/*                                     Make sure that this Access Module supports the close type */

if ( Opts->Reqtype == pmiPIDMOptCloser ) retcode = pmrcUnsupported;
else
{
    fclose (Fp);
    Fp = 0;
}
break;
/*-----*/
/*                                     Read */
/*-----*/
case pmiPIDMOptRead:

```

Appendix B: Access Module Examples  
Sample Program

```
/*                                Make sure that the file has been opened */
    if ( !Fp )
    {
        strcpy (SG_ErrorText,"Attempt to read, no File Open");
        SG_ErrorTextLen = strlen(SG_ErrorText);
        retcode = pmrcFailure;
    }
    if ( retcode ) break;

/*                                Keep track of the position of the most recent block read */
/*                                for possible get position request (via Reqtype pmiPIDMOptGetPos) */

    FileOffset = ftell(Fp);

/*                                read data DIRECTLY into the buffer provided by the DIAM */
    frc = fread ( ((pmiRW_t*)OptParms)->Buffer,1,AXSM_DEFAULT_BLKSIZE,Fp);

/*                                Check for any I/O errors */
/*                                Trap End-of-file condition */
    if ( frc < 1 )
    {
        if ( feof(Fp) ) retcode = pmrcEOF;
        else
        {
            strcpy (SG_ErrorText,"Unknown fread error");
            SG_ErrorTextLen = strlen(SG_ErrorText);
            retcode = pmrcFailure;
        }
        frc = 0;
    }

/*                                Return the record length */
    ((pmiRW_t*)OptParms)->BufferLen = (pmUInt32) frc;
    break;
/*-----*/
/*                                Write */
/*-----*/
    case pmiPIDMOptWrite:

/*                                Make sure that the file has been opened */
    if ( !Fp )
    {
        strcpy (SG_ErrorText,"Attempt to write, no File Open");
        SG_ErrorTextLen = strlen(SG_ErrorText);
        retcode = pmrcFailure;
    }
    if ( retcode ) break;

/*                                Check for non-zero record length and write */
```

```

if ( ((pmiRW_t*)OptParms)->BufferLen)
{
    amtWritten = fwrite( ((pmiRW_t*)OptParms)->Buffer, 1,
                        ((pmiRW_t*)OptParms)->BufferLen, Fp);
    if (amtWritten != ((pmiRW_t*)OptParms)->BufferLen)
    {
        strcpy (SG_ErrorText, "Unknown fwrite error");
        SG_ErrorTextLen = strlen(SG_ErrorText);
        retcode = pmrcFailure;
    }
}

/*          If write OK, get the current position for possible get */
/*          position request (via Reqtype pmiPIDMOptGetPos) */

    if ( !retcode) FileOffset = ftell(Fp);
    break;

/*-----*/
/*          Get Media Position */
/*-----*/
    case pmiPIDMOptGetPos:

/*          Make sure that the file has been opened */

    if ( !Fp )
    {
        strcpy (SG_ErrorText, "Attempt to get position, no File Open");
        SG_ErrorTextLen = strlen(SG_ErrorText);
        retcode = pmrcFailure;
    }
    if ( retcode ) break;

/*          Pick up the position info that we keep of the last block accessed */

    ((pmiPos_t*)OptParms)->Position.Length = sizeof(FileOffset);
    ((pmiPos_t*)OptParms)->Position.Data = (char *) &FileOffset;
    break;

/*-----*/
/*          Set Media Position */
/*-----*/
    case pmiPIDMOptSetPos:

/*          Make sure that the file has been opened */

    if ( !Fp )
    {
        strcpy (SG_ErrorText, "Attempt to set position, no File Open");
        SG_ErrorTextLen = strlen(SG_ErrorText);
        retcode = pmrcFailure;
    }
    if ( retcode ) break;

```

**Sample Program**

```

/*          Make sure the provided buffer is the same size as the */
/*          Access Module expects */

    if ( ((pmiPos_t*)OptParms)->Position.Length != sizeof(FileOffset))
        {
            retcode = pmrcPosDataInvL;
        }
    if ( retcode ) break;

/*          Copy the provided position info into an appropriate data type */
/*          and do the reposition operation */

    memcpy(&TmpOffset, ((pmiPos_t*)OptParms)->Position.Data,
           sizeof(TmpOffset));

    seekrc = fseek(Fp, TmpOffset, 0);
    if (seekrc)
        {
            strcpy (SG_ErrorText, "Unknown fseek error");
            SG_ErrorTextLen = strlen(SG_ErrorText);
            retcode = pmrcFailure;
        }
    else FileOffset = TmpOffset;

    break;

/*-----*/
/*          Get and Put Access Module or File attributes */
/*-----*/
    case pmiPIDMOptGetA_A:
        if ( strcmp( ((pmiAttr_t*)OptParms)->AttrName, "AXMAttribute#1") )
            retcode = pmrcBadAttrName;
        else
            {
                strcpy( &((pmiAttr_t*)OptParms)->AttrValue[0], AXMAttr_1);
                ((pmiAttr_t*)OptParms)->AttrValueLen = strlen(AXMAttr_1);
            }
        break;
    case pmiPIDMOptGetF_A:
        if ( strcmp( ((pmiAttr_t*)OptParms)->AttrName, "FileAttribute#1") )
            retcode = pmrcBadAttrName;
        else
            {
                strcpy( &((pmiAttr_t*)OptParms)->AttrValue[0], FileAttr_1);
                ((pmiAttr_t*)OptParms)->AttrValueLen = strlen(FileAttr_1);
            }
        break;
    case pmiPIDMOptPutA_A:
        if ( strcmp( ((pmiAttr_t*)OptParms)->AttrName, "AXMAttribute#1") )
            retcode = pmrcBadAttrName;

```

```

        break;
    case pmiPIDMOptPutF_A:
        if ( strcmp( ((pmiAttr_t*)OptParms)->AttrName,"FileAttribute#1") )
            retcode = pmrcBadAttrName;
        break;
/*-----*/
/*                                     Access Module shutdown */
/*-----*/
    case pmiPIDMOptShut:
/*                                     If any files are currently open, close them */
        if ( Fp )
        {
            fclose(Fp);
            Fp = 0;
        }
        retcode = pmrcOK;
        break;
/*-----*/
/*                                     Access Module identification */
/*-----*/
    case pmiPIDMOptID:
/*                                     Supply the single module name for this Access Module identification */
/*                                     If more modules are added, their identifying information */
/*                                     must be appended to the list via VerIDList.Next */

        strcpy( ((pmiID_t*)OptParms)->VerIDList.ModuleName,MODULE_NAME);
        strcpy( ((pmiID_t*)OptParms)->VerIDList.ModuleVers,MODULE_VERS);
        ((pmiID_t*)OptParms)->VerIDList.Next = 0;
        break;
/*-----*/
/*                                     Eh? */
/*-----*/
    default:
/*                                     Handle unknown request type */

        retcode = pmrcUnsupported;
    } /* end switch on request type */

/*-----*/
/*                                     Whatever action was to be taken has been taken */
/*                                     General overhead before returning to the DIAM */
/*-----*/

/*                                     Insert the established return code into the command block */
/*                                     for return to the DIAM */

    Opts->Retcode = retcode;

/*     If there's a Access Module specific error, indicated by a return code */

```

**Sample Program**

```
/*          of pmrcFailure, insert the associated error text into */
/*          the command block for return to the DIAM */

if ( retcode == pmrcFailure )
{
    Opts->ErrMsg.Data = &SG_ErrorText[0];
    Opts->ErrMsg.DataLength = SG_ErrorTextLen;
}
else
{
    Opts->ErrMsg.Data = 0;
    Opts->ErrMsg.DataLength = 0;
}

return;
}
```

---

# Sample Makefile

Before running make, set the symbol INCDIR to the directory in which pmddamt.h and pmdcomt.h are stored. Set the symbol SRCDIR to the directory in which the sample source (or yours) is stored.

```
OBMS      = EGaxssm.o
INCDIR    = ../inc
SRCDIR    = ../samples
NESTINC   = $(INCDIR)/pmdcomt.h \
            $(INCDIR)/pmddamt.h
CCFlags   = -G -KPIC

EGaxsm.o: $(NESTINC) $(SRCDIR)/EGaxsm.c
          cc $(CCFlags) $(SRCDIR)/EGaxsm.c
# DO NOT DELETE THIS LINE -- make depends on it.
```



# Index

---

## A

- Access modules
  - attribute exchange functions 2-3
  - definition 1-2
  - functional description 2-2
  - functions
    - file close 3-13
    - file get position 3-19
    - file open 3-10
    - file read 3-15
    - file set position 3-21
    - file write 3-17
    - general function-independent structure 3-2
    - get attribute 3-25
    - identification 3-8
    - initialization 3-5
    - interface return codes 3-4
    - put attribute 3-23
    - request type function requests 3-3
    - shutdown 3-27
  - get functions 2-3
  - interface 2-2
  - linkages, diagram 1-2
  - main function 2-2
  - programming considerations 3-1
  - put functions 2-4
- Attribute exchange functions, access modules 2-3
- AttrName parameter
  - get attribute function 3-25
  - put attribute function 3-23
- AttrNameLen parameter
  - get attribute function 3-25
  - put attribute function 3-23
- AttrValue parameter
  - get attribute function 3-25
  - put attribute function 3-23
- AttrValueLen parameter
  - get attribute function 3-25
  - put attribute function 3-23

---

## B

- BlkHeaderLen parameter, file open function 3-11
- BlkSize parameter, file open function 3-11
- Buffer parameter
  - file read function 3-15
  - file write function 3-17

- BufferLen parameter
  - file read function 3-15
  - file write function 3-17

---

## C

- character sets 3-24
- CHARSET\_NAME 3-24
- CHARSET\_NUMBER 3-24
- ClientID parameter, initialization function 3-6
- ClientIDL parameter, initialization function 3-6

---

## D

- Data Connector API, definition 1-2
- documentation
  - related to Access Module ii

---

## E

- ErrMsg parameter, general function-independent structure 3-3
- EyeCatcher parameter
  - file close function 3-13
  - file get position function 3-19
  - file open function 3-10
  - file read function 3-15
  - file set position function 3-21
  - file write function 3-17
  - general function-independent structure 3-2
  - get attribute function 3-25
  - identification function 3-8
  - initialization function 3-5
  - put attribute function 3-23
  - shutdown function 3-27

---

## F

- FIDData parameter
  - file close function 3-13
  - file get position function 3-19
  - file open function 3-11

file read function 3–15  
 file set position function 3–21  
 file write function 3–17  
 File close function, access modules 3–13  
 File get position function, access modules 3–19  
 File open function, access modules 3–10  
 File read function, access modules 3–15  
 File set position function, access modules 3–21  
 File write function, access modules 3–17  
 FileName parameter, file open function 3–11  
 FileNameL parameter, file open function 3–11

---

## G

General function-independent structure, access modules 3–2  
 Get attribute function, access modules 3–25  
 Get functions, access modules 2–3

---

## I

Identification function, access modules 3–8  
 Initialization function, access modules 3–5  
 InitStr parameter, initialization function 3–6  
 InitStrL paramete,  
   initialization function 3–6  
 Interface functions, access modules 2–2  
 Interface return codes, access modules 3–4  
 InterfaceVerNoD parameter, initialization function 3–5  
 InterfaceVerNoD parameter, initialization function 3–6

---

## M

Main function, access modules 2–2  
 Mode parameter  
   shutdown function 3–27

---

## O

ObjData parameter  
   get attribute function 3–25  
   put attribute function 3–23  
 OpenMode parameter, file open function 3–11

---

## P

PIData parameter  
   file open function 3–10  
   initialization function 3–5  
   shutdown function 3–27  
 pmrcAllocErr return code, general  
   function-independent structure 3–4  
 pmrcAPINotReady return code, general  
   function-independent structure 3–4  
 pmrcBadAttrName return code  
   get attribute function 3–26  
   put attribute function 3–24  
 pmrcBadFormat return code, file open function 3–11  
 pmrcBadFp return code  
   file close function 3–13  
   file read function 3–15  
   file write function 3–17  
 pmrcBadParm return code, general  
   function-independent structure 3–4  
 pmrcBadTraceLvl return code, general  
   function-independent structure 3–4  
 pmrcBadVer return code, initialization function 3–6  
 pmrcEOF return code, file read function 3–15  
 pmrcFailure return code, general  
   function-independent structure 3–4  
 pmrcFNameTooLong return code, file open  
   function 3–11  
 pmrcOK return code, general function-independent  
   structure 3–4  
 pmrcOpenFiles return code, shutdown function 3–27  
 pmrcReadOnly return code, file write function 3–17  
 pmrcRedundant return code, initialization  
   function 3–6  
 pmrcUnsupported return code, general  
   function-independent structure 3–4  
 pmrcWriteOnly return code, file read function 3–15  
 Position parameter  
   file get position function 3–19  
   file set position function 3–21  
 prerequisites i  
 Programming considerations, access modules 3–1  
 Put attribute function, access modules 3–23  
 Put functions, access modules 2–4

---

## R

Record formats 1–2  
   table of 1–3  
 Reqtype parameter, general function-independent  
   structure 3–2

Request type function requests, access modules 3-3

Retcoder parameter, general function-independent structure 3-2

Return codes

- pmrcAllocErr 3-4
- pmrcAPInotReady 3-4
- pmrcBadAttrName 3-24, 3-26
- pmrcBadFormat 3-11
- pmrcBadFp 3-13, 3-15, 3-17
- pmrcBadParm 3-4
- pmrcBadTraceLvl 3-4
- pmrcBadVer 3-6
- pmrcEOF 3-15
- pmrcFailure 3-4
- pmrcFNameTooLong 3-11
- pmrcOK 3-4
- pmrcOpenFiles 3-27
- pmrcReadOnly 3-17
- pmrcRedundant 3-6
- pmrcUnsupported 3-4
- pmrcWriteOnly 3-15

---

## S

Shutdown function, access modules 3-27

software releases, supported i

StructLength parameter

- file close function 3-13
- file get position function 3-19
- file open function 3-10
- file read function 3-15
- file set position function 3-21
- file write function 3-17
- general function-independent structure 3-2
- get attribute function 3-25
- identification function 3-8
- initialization function 3-5
- put attribute function 3-23
- shutdown function 3-27

---

## T

TraceLvl parameter, general function-independent structure 3-2

---

## V

VerIDList parameter, identification function 3-8

