Dialogic
Making Innovation Thrive™

Dialogic® NaturalAccess™ Digital Trunk Monitoring API Developer's Manual

October 2009 64-0500-01

www.dialogic.com

Copyright and legal notices

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation or its subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Due to differing national regulations and approval requirements, certain Dialogic products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Dialogic Corporation at the address indicated below or on the web at www.dialogic.com.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.

Any use case(s) shown and/or described herein represent one or more examples of the various ways, scenarios or environments in which Dialogic® products can be used. Such use case(s) are non-limiting and do not represent recommendations of Dialogic as to whether or how to use Dialogic products.

Dialogic, Dialogic Pro, Brooktrout, Diva, Cantata, SnowShore, Eicon, Eicon Networks, NMS Communications, NMS (stylized), Eiconcard, SIPcontrol, Diva ISDN, TruFax, Exnet, EXS, SwitchKit, N20, Making Innovation Thrive, Connecting to Growth, Video is the New Voice, Fusion, Vision, PacketMedia, NaturalAccess, NaturalCallControl, NaturalConference, NaturalFax and Shiva, among others as well as related logos, are either registered trademarks or trademarks of Dialogic Corporation or its subsidiaries. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement.

Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries. The names of actual companies and product mentioned herein are the trademarks of their respective owners.

This document discusses one or more open source products, systems and/or releases. Dialogic is not responsible for your decision to use open source in connection with Dialogic products (including without limitation those referred to herein), nor is Dialogic responsible for any present or future effects such usage might have, including without limitation effects on your products, your business, or your intellectual property rights.

Revision history

Revision	Release date	Notes
9000-6392-10B	August, 1997	Beta release, CYF/ERS
9000-6392-11	October, 1998	ERS
9000-6392-12	February, 1999	SRG
9000-6392-13	July, 2000	CT Access 3.0/4.0 LBG
9000-6392-14	March, 2001	SRR, NACD 2000-2
9000-6392-15	April, 2001	LBG, NACD 2001-1
9000-6392-16	June, 2001	LBG
9000-6392-17	November, 2001	SRR, NACD 2002-1 Beta
9000-6392-18	May, 2002	MCM, NACD 2002-1
9000-6392-19	April, 2003	LBG, Natural Access 2003-1
9000-6392-20	April, 2004	MCM, Natural Access 2004-1
64-0500-01	October 2009	LBG, NaturalAccess R9.0
Last modified: September 3, 2009		

Refer to the www.dialogic.com for product updates and for information about support policies, warranty information, and service offerings.

Table Of Contents

1 Introduction

This manual describes the Digital Trunk Monitoring (DTM) API. The DTM API enables NaturalAccess applications to monitor for alarms on digital trunks and to gather performance statistics.

This document is intended for developers of telephony and voice applications who are using NaturalAccess. This document defines telephony terms where applicable, but assumes that you are familiar with telephony concepts and the C programming language.

2 Terminology

Note: The product to which this document pertains is part of the NMS Communications Platforms business that was sold by NMS Communications Corporation ("NMS") to Dialogic Corporation ("Dialogic") on December 8, 2008. Accordingly, certain terminology relating to the product has been changed. Below is a table indicating both terminology that was formerly associated with the product, as well as the new terminology by which the product is now known. This document is being published during a transition period; therefore, it may be that some of the former terminology will appear within the document, in which case the former terminology should be equated to the new terminology, and vice versa.

Former terminology	Dialogic terminology
CG 6060 Board	Dialogic® CG 6060 PCI Media Board
CG 6060C Board	Dialogic® CG 6060C CompactPCI Media Board
CG 6565 Board	Dialogic® CG 6565 PCI Media Board
CG 6565C Board	Dialogic® CG 6565C CompactPCI Media Board
CG 6565e Board	Dialogic® CG 6565E PCI Express Media Board
CX 2000 Board	Dialogic® CX 2000 PCI Station Interface Board
CX 2000C Board	Dialogic® CX 2000C CompactPCI Station Interface Board
AG 2000 Board	Dialogic® AG 2000 PCI Media Board
AG 2000C Board	Dialogic® AG 2000C CompactPCI Media Board
AG 2000-BRI Board	Dialogic® AG 2000-BRI Media Board
NMS OAM Service	Dialogic® NaturalAccess™ OAM API
NMS OAM System	Dialogic® NaturalAccess™ OAM System
NMS SNMP	Dialogic® NaturalAccess™ SNMP API
Natural Access	Dialogic® NaturalAccess™ Software
Natural Access Service	Dialogic® NaturalAccess™ Service
Fusion	Dialogic® NaturalAccess™ Fusion™ VoIP API
ADI Service	Dialogic® NaturalAccess™ Alliance Device Interface API
CDI Service	Dialogic® NaturalAccess™ CX Device Interface API
Digital Trunk Monitor Service	Dialogic® NaturalAccess™ Digital Trunk Monitoring API
MSPP Service	Dialogic® NaturalAccess™ Media Stream Protocol Processing API
Natural Call Control Service	Dialogic® NaturalAccess™ NaturalCallControl™ API
NMS GR303 and V5 Libraries	Dialogic® NaturalAccess™ GR303 and V5 Libraries

Former terminology	Dialogic terminology
Point-to-Point Switching Service	Dialogic® NaturalAccess™ Point-to-Point Switching API
Switching Service	Dialogic® NaturalAccess™ Switching Interface API
Voice Message Service	Dialogic® NaturalAccess™ Voice Control Element API
NMS CAS for Natural Call Control	Dialogic® NaturalAccess™ CAS API
NMS ISDN	Dialogic® NaturalAccess™ ISDN API
NMS ISDN for Natural Call Control	Dialogic® NaturalAccess™ ISDN API
NMS ISDN Messaging API	Dialogic® NaturalAccess™ ISDN Messaging API
NMS ISDN Supplementary Services	Dialogic® NaturalAccess™ ISDN API Supplementary Services
NMS ISDN Management API	Dialogic® NaturalAccess™ ISDN Management API
NaturalConference Service	Dialogic® NaturalAccess™ NaturalConference™ API
NaturalFax	Dialogic® NaturalAccess™ NaturalFax™ API
SAI Service	Dialogic® NaturalAccess™ Universal Speech Access API
NMS SIP for Natural Call Control	Dialogic® NaturalAccess™ SIP API
NMS RJ-45 interface	Dialogic® MD1 RJ-45 interface
NMS RJ-21 interface	Dialogic® MD1 RJ-21 interface
NMS Mini RJ-21 interface	Dialogic® MD1 Mini RJ-21 interface
NMS Mini RJ-21 to NMS RJ-21 cable	Dialogic® MD1 Mini RJ-21 to MD1 RJ-21 cable
NMS RJ-45 to two 75 ohm BNC splitter cable	Dialogic® MD1 RJ-45 to two 75 ohm BNC splitter cable
NMS signal entry panel	Dialogic® Signal Entry Panel

3 Digital Trunk Monitor service overview

Digital Trunk Monitor service definition

The Digital Trunk Monitor (DTM) service is a Natural Access service. Natural Access is a development environment for computer-based telephony applications that provides a standard, hardware-independent programming interface.

A Natural Access service is a group of logically related telephony functions. Each service is contained in a service manager, which is a software wrapper surrounding one or more services. The service manager implements binding functions that enable a service to integrate with Natural Access.

A trunk is a transmission channel between two switching stations. For the purpose of the DTM service, consider a BRI, T1, or E1 line (the aggregate of 2, 24, or 30 channels) a single trunk.

The DTM service enables an application to:

- Monitor for alarms on T1 and E1 trunks
- Retrieve trunk state information on BRI trunks
- Gather performance statistics
- Easily detect digital trunk failures

The DTM service can be used to collect statistics and report error counts. A DTM event is generated when any count changes. The application can synchronously retrieve the current counter values from the DTM service's internal cache.

The DTM service demonstration program *trunklog* is included with Natural Access.

Alarms and digital trunks

A T1 trunk enters an alarm state upon the presence of red, yellow, or blue alarms. An E1 trunk enters an alarm state upon receipt of local or remote loss of frame errors or excessive bit errors. When the DTM service is used as an alarm monitor, an event is generated when a digital trunk goes into or out of an alarm state. Since failure monitoring is done in the same processing context as call processing, an application can terminate a call and mark its port as unavailable upon receipt of a trunk failure event.

Multiple trunks can be monitored using a single context. The same trunk can be monitored by multiple contexts (and multiple processes).

Note: The alarm state is not applicable to BRI trunks.

Natural Access environment

This topic describes:

- The Natural Access programming model
- The Natural Access components
- Managing parameters in Natural Access

For more detailed information about Natural Access and the ADI service, see the *Natural Access Developer's Reference Manual* and the *ADI Service Developer's Reference Manual*.

Programming model

Natural Access employs an asynchronous programming model to take advantage of concurrent processing. Each function called returns immediately with either a return value of SUCCESS or an error code. Synchronous functions are complete when the return code is received.

For asynchronous operations, if the return value is SUCCESS, the function is successfully initiated, and the execution result arrives asynchronously. If the return code is not SUCCESS, the operation is not initiated and no events associated with the particular function are generated.

The execution result is indicated by an event or a series of events. An event is a data structure with an event ID that identifies an operation and operation-specific results. For more information about events in the Natural Access environment, refer to the *Natural Access Developer's Reference Manual*.

Natural Access provides common system error codes that all the services use. These error codes are defined in the header file *ctaerr.h* and have a CTAERR_ prefix. Refer to the *Natural Access Developer's Reference Manual* for detailed descriptions of Natural Access error codes, and to the *ADI Service Developer's Reference Manual* for detailed descriptions of ADI service error codes. Use **ctaGetText** to obtain the text description of an error code.

Natural Access components

Natural Access organizes services and accompanying resources around a single processing unit called a context. To access service functionality, an application creates a context and attaches the services it requires. Only one instance of a service can be opened on a single context.

A context usually represents an application thread performing a related set of functions, such as controlling a single line. The context maintains defined parameters for each type of service, allowing each line to have its own characteristics.

Some contexts are not associated with a line. For example, an operation performing speech recognition on a buffer of data does not require a telephone line.

To use the DTM service in the Natural Access environment, an application must obtain a context and open the DTM service on it.

An event queue is the communication path from a service to an application. A service generates events indicating certain conditions or state changes. An application retrieves the events from the event queue.

When the event queue is created, you specify the service managers to be attached to it. Use the ADI service manager (ADIMGR) for the DTM service. When creating a context, you specify an associated event queue. All events from the services on the context are sent to the event queue attached to the context.

One queue can be created for the entire application process or multiple queues can be used. The number of queues you create depends on the programming model used. Some models use one queue to handle all contexts, and handle multiple telephone lines through the same queue. Other models use one queue for each telephone line.

Managing parameters in Natural Access

Natural Access service characteristics can be altered by modifying associated parameters. Each parameter structure has default values that are sufficient for most applications.

Natural Access manages parameters for services on a context basis. The context maintains a copy of the parameters for all services opened on the context. This allows each operation to have its own characteristics.

The following Natural Access functions allow you to obtain or change parameter information:

- **ctaGetParmByName** retrieves a single field for a given parameter name.
- **ctaSetParmByName** modifies a single field for a given parameter name.
- **ctaGetParmInfo** retrieves a parameter field definition.
- **ctaGetParms** returns parameter values for a given parameter structure.
- **ctaRefreshParms** resets the values of all context parameters on a context to the global defaults.

Refer to the *Natural Access Developer's Reference Manual* for details about Natural Access parameter management and Natural Access functions.

4 Using the DTM service

Setting up the Natural Access environment

The Digital Trunk Monitor service is a C function library component of Natural Access. It is implemented as the DTM service and managed by the ADI service manager.

The DTM service is a DLL under Windows and a shared library under UNIX. You must have Natural Access, including the ADI service, installed on your system to build and run applications using the DTM service.

To set up a Natural Access application:

1. Initialize the Natural Access application.
2. Create event queues.
3. Create contexts for each event queue.
4. Open services on each context.

To set up one or more additional Natural Access applications that share a context with the first application:

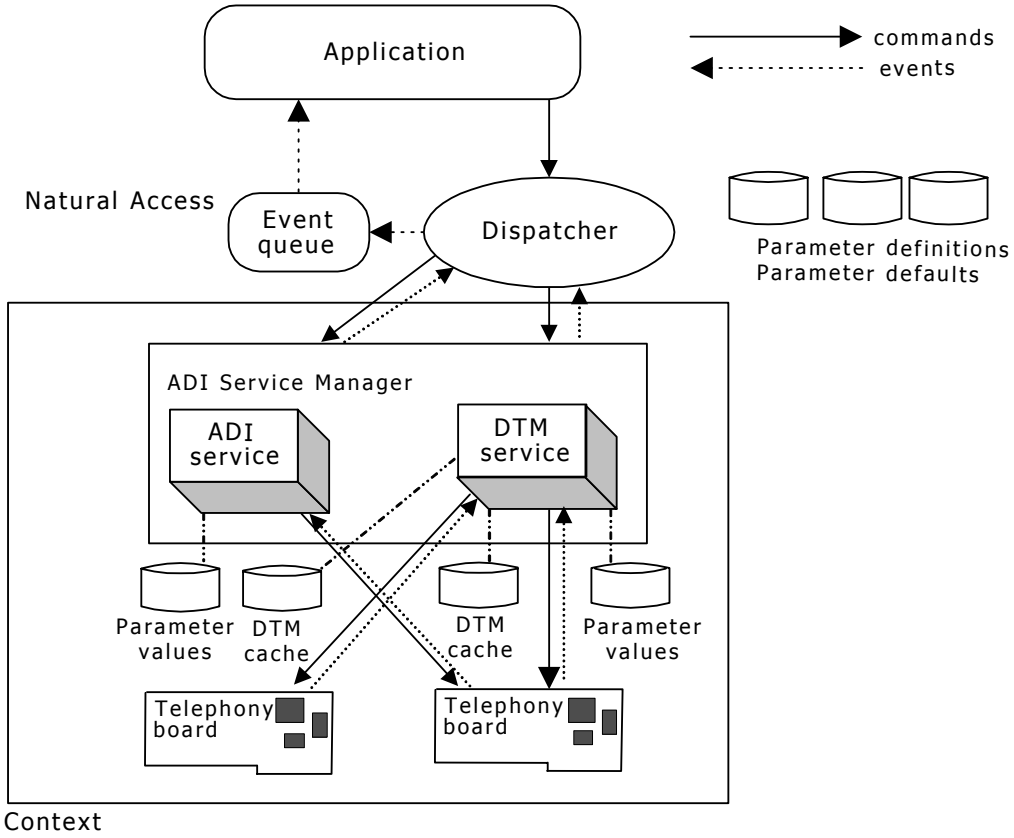
1. Initialize the Natural Access application.
2. Create event queues.
3. Attach to the existing context that the applications will share.

Note: For more information on setting up applications, refer to the *Natural Access Developer's Reference Manual.*

Services are opened on a context by calling **ctaOpenServices**, passing a context handle and a list of service descriptors. Each service descriptor specifies the name of the service, service manager, and service-specific arguments.

The call to **ctaOpenServices** is asynchronous and returns immediately. When all services are open, CTAEVN_OPEN_SERVICES_DONE is returned to the application. Natural Access supports opening and closing services on an as-needed basis to share scarce resources.

There can be only one open instance of each service per context. Once the DTM service is open, the application functions invoke the DTM service. The following illustration shows an example of a context with the DTM and ADI services open. The DTM service is associated with trunks on two telephony boards.



Natural Access with the ADI service and the DTM service

ctaCloseServices closes one or more services on a context. The function is asynchronous and returns immediately. You must wait for the completion event CTAEVN_CLOSE_SERVICES_DONE to be returned on the event queue before you can assume that the specified services are closed. An application can close or re-open one service at a time without affecting other open services.

ctaDestroyContext destroys a context. All currently opened services on the context are closed. The function is asynchronous and returns immediately. You must wait for CTAEVN_DESTROY_CONTEXT_DONE to be returned on the event queue before assuming that the context is destroyed and resources are released.

ctaDestroyQueue destroys the queue and all contexts associated with the queue. This is a synchronous function, so the application remains blocked until all clean up and close activity is completed. If the services and contexts are already closed, the function returns immediately.

Development environment

To use the DTM service functions, an application must link to the following files:

- DTM service header file *dtmdef.h*.
- Import library *adidtm.lib* (named *libadidtm.so* under UNIX).
- DTM service Windows dynamic link library *adidtm.dll* or the UNIX shared library *libadidtm.so*.

Looking at the annotated source code for the demonstration programs is a good way to start developing your own applications. The demonstration programs are installed in the *\nms\ctaccess\demos* directory (*/opt/nms/ctaccess/demos* under UNIX). Each demonstration program has its own subdirectory with the same name as the demonstration program.

Note: The directory structure can vary slightly according to your operating system and installation. Refer to the *Natural Access Developer's Reference Manual* for specific information about the appropriate text file to examine to retrieve the exact path to the header and library files on your system.

Setting up the DTM service

Before you can call functions from the DTM service, the application must initialize and open the DTM service on an open context. This topic presents:

- Initializing the DTM service
- Opening the DTM service

Initializing the DTM service

To initialize the DTM service, include the DTM service and the ADI Service Manager in the call to **ctaInitialize**.

The following code excerpt demonstrates initializing the DTM service together with the ADI and Voice Message services:

```
void MyServiceInit()
    {
        DWORD ret;
        CTA_INIT_PARMS initparms = { 0 };
        CTA_ERROR_HANDLER hdlr;
        CTA_SERVICE_NAME InitServices[] =    /* for ctaInitialize */
        {  { "ADI", "ADIMGR" },
           { "DTM", "ADIMGR" },
           { "VCE", "VCEMGR" },
        };
        /* Initialize size of init parms structure */
        initparms.size = sizeof(CTA_INIT_PARMS);

        if ( ( ret = ctaInitialize(
                         InitServices,
                         sizeof(InitServices)/sizeof(InitServices[0]),
                         &initparms)) != SUCCESS)
        {
        /* ... handle error conditions here.... */

        }
    }
```

After the call to **ctaInitialize**:

1. Create an event queue attached to the ADI Service Manager by calling **ctaCreateQueue**. ADIMGR can be explicitly specified in the call. If NULL is passed, all service managers specified in **ctaInitialize** are attached.
2. Create a context by calling **ctaCreateContext**.

Opening the DTM service

The DTM service must be opened on a context in order to use DTM service functions. Opening the DTM service starts the trunk monitor software on the board and enables it to make status information accessible to the application at specific intervals.

Note: Only one instance of the DTM service can be opened on each context.

When the application opens a service, it specifies a board number. When the application opens the trunk monitor, it binds the board to a context.

To open the DTM service on a context, call **ctaOpenServices**. This function takes an array of CTA_SERVICE_DESC structures as an input argument. Each CTA_SERVICE_DESC structure defines a service (in this case, the DTM service). When a service is opened successfully, CTAEVN_OPEN_SERVICES_DONE with CTA_REASON_FINISHED is delivered to the application. The CTA_SERVICE_DESC structure is defined as follows:

```
typedef struct CTA_SERVICE_DESC
{
    CTA_SERVICE_NAME name;    /* service name                        */
    CTA_SERVICE_ADDR svcaddr; /* reserved                            */
    CTA_SERVICE_ARGS svcargs; /* passes service-specific arguments   */
    CTA_MVIP_ADDR mvipaddr;   /* AG board #, stream, timeslot, mode */

}CTA_SERVICE_DESC;
```

Note: The DTM service does not take any information from the MVIP_ADDR structure. You specify which board, stream, and timeslot to monitor in a DTM function call when you start the trunk monitor, not when you open the DTM service. The CTA_SERVICE_ARGS structure is not used by the DTM service.

The following code sample demonstrates how an application opens the DTM service:

```
DWORD ret ;
CTA_EVENT event ;
CTA_SERVICE_DESC service[] =
{
   { {"DTM", "ADIMGR"}, { 0 }, { 0 }, { 0 } } ,
} ;

                                /* open the DTM service           */
ret = ctaOpenServices(  ctahd,  /* a context handle               */
                        services,
                        1 );
if(ret != SUCCESS)
{
                                /* opening DTM service failed     */
    DemoShowError( "ctaOpenServices", ret );
}
else
{
                   /* wait for the CTAEVN_OPEN_SERVICES_DONE event */
     DemoWaitForSpecificEvent(ctahd,
                                CTAEVN_OPEN_SERVICES_DONE,
                                & event ) ;

                                /* check the reason of completion */
     if (event.value != CTA_REASON_FINISHED)
     {
         DemoShowError( "ctaOpenServices", event.value );
     }
}
```

Monitoring a digital trunk

This topic presents:

- Trunk monitor objects
- Retrieving trunk status information
- Cooperative trunk monitoring
- Generating alarms

The DTM service maintains accumulator fields for T1/E1 trunks. It can monitor the following conditions:

Condition	Description
Errored seconds	One second intervals containing one or more of the following errors: • Loss of frame • Framing error • Bipolar violation • AIS (all ones) • Slip • E1 CRC error
Severely errored seconds	One-second intervals in which the bit error rate exceeds 10^{-3}, or an out-of-frame error occurred.
Unavailable seconds	One-second intervals. On a T1 trunk, they are preceded by 10 consecutive severely errored seconds. On an E1 trunk, loss of signal occurred, out-of-frame occurred, or excessive bit error rate was detected. Unavailable seconds also indicate a failure state that includes any of the following conditions: • Far end alarm (yellow alarm on T1) • Loss of frame (red alarm on T1) • AIS (all ones) • Loss of signal

For BRI trunks, the DTM service can also maintain accumulator fields for statistics and trunk state. It can monitor the following conditions:

- Slip
- Framing errors
- Received and transmitted frames
- Trunk state
- B channel status

To start monitoring a digital trunk, call **dtmStartTrunkMonitor**. **dtmStartTrunkMonitor** takes a pointer to DTM_START_PARMS as one of its arguments. The maxevents field in DTM_START_PARMS determines the maximum number of events that can be sent from the board per second. The reportmask field in the DTM_START_PARMS structure specifies which of the following conditions the monitor reports:

Name	Value	Condition monitored
DTM_REPORT_GO_NOGO	1	Going in or out of alarm state.
DTM_REPORT_ALARMS	2	Any change in received alarms.
DTM_REPORT_STATUS	4	Any status or synchronization change.
DTM_REPORT_SLIPS	8	Any slip.
DTM_REPORT_ES	0x10	Change in errored seconds.
DTM_REPORT_SES	0x20	Change in severely errored seconds.
DTM_REPORT_UAS	0x40	Change in unavailable seconds.

Call **dtmStopTrunkMonitor** to terminate the trunk monitor.

Trunk monitor objects

Calling **dtmStartTrunkMonitor** returns a DTM handle, ***dtmhd***. The DTM handle uniquely identifies a trunk monitor object that represents the board-trunk pair that is being monitored. Only one board can be associated with a trunk monitor object. Create multiple trunk monitor objects to monitor multiple trunks. Use multiple trunk monitor objects to monitor multiple boards. An application can monitor as many trunks per context as it needs, and report events from each of those trunks on a single event queue associated with the context. The ***dtmhd*** is the unique identifier for each trunk.

A ***dtmhd*** is valid until a **dtmStopTrunkMonitor** function call stopping the monitor on the specified trunk has completed and DTMEVN_MONITOR_DONE is returned to the application.

Retrieving trunk status information

The DTM service maintains a cache on the host computer for each trunk, where the trunk's status is stored. Call **dtmGetTrunkStatus** (or **dtmGetBriTrunkStatus**) to retrieve a snapshot of a specified trunk's status. DTM_TRUNK_STATUS (or DTM_BRI_TRUNK_STATUS) contains current information about alarms, trunk status, and error statistics. The same structure (DTM_TRUNK_STATUS) is used for both T1 and E1 trunks.

Complete the following steps to retrieve the trunk status:

Step	Action
1	Call **dtmRefreshTrunkStatus** to force the board to refresh the cache on the host with the most recent information about the specified trunk. The board sends a trunk status event immediately upon receiving this request.
2	Call **dtmResetCounters** to request that the board reinitialize the accumulator and start time fields and force a status event to be sent to the host from the board, which also refreshes the DTM_TRUNK_STATUS (or DTM_BRI_TRUNK_STATUS) structure. See DTM_TRUNK_STATUS structure for more details about the DTM_TRUNK_STATUS (or DTM_BRI_TRUNK_STATUS) structure.

Cooperative trunk monitoring

Natural Access enables two or more applications to cooperate in the monitoring of a trunk using a shared context and a shared trunk monitor object. For example, one application can start trunk monitoring and another application can stop trunk monitoring.

To do so, one application must first create the trunk monitor object by invoking **dtmStartTrunkMonitor**. Then other applications must obtain a handle to that object in one of the following ways:

- The first application calls **ctaGetObjDesc** twice and obtains two descriptors: one for the context and the other for the trunk monitor object. The application passes both descriptors to the second application. The second application calls **ctaAttachContext** to attach to the context and invokes **ctaAttachObject** to obtain a ***dtmhd*** for the attached trunk monitor object.

  For more information on descriptors, refer to the *Natural Access Developer's Reference Manual*.

- The first application calls **ctaGetObjDesc** and obtains a context descriptor for the context associated with the trunk monitor object. The application passes the board ID, trunk ID, and context descriptor to the other application.

  The other application then calls **ctaAttachContext** to obtain a context handle.

  The application invokes **dtmAttachTrunkMonitor** to attach to the trunk monitor object.

- The first application calls **ctaGetObjDesc** and obtains a context descriptor for the context associated with the trunk monitor object. The application passes the context descriptor to the other application.

  The other application then calls **ctaAttachContext** to obtain a context handle.

  The application receives an event containing the ***dtmhd*** in the objhd field of the CTA_EVENT structure.

In a context-sharing environment, DTMEVN_MONITOR_DONE is sent to all applications sharing a context. Upon receiving this event, the ***dtmhd*** is released and is no longer valid.

Generating alarms

Call **dtmSendAlarm** to start sending remote alarms on the specified trunk, or to turn off any specific configuration to send remote alarms. The remote alarms are what the other end of the trunk sees, and are not necessarily the same alarms that are sent from the board to the application. Any alarms that are automatically generated are unaffected by **dtmSendAlarm**. **dtmSendAlarm** accepts the following values for its alarm argument:

Name	Value	Alarm
DTM_SEND_NO_ALARMS	0	Stop sending alarms (automatic alarms can still occur).
DTM_SEND_AIS	1	Send AIS (all ones signal - blue alarms).
DTM_SEND_LOF	2	Send loss of frame (yellow alarm).
DTM_SEND_TS16AIS	3	Send TS16AIS (E1 only).

Note: BRI trunks do not support **dtmSendAlarm**.

5 Function reference

Function summary

The DTM service provides the following functions:

Function	Synchronous/ Asynchronous	Description
dtmAttachTrunkMonitor	Synchronous	Enables an application to attach to a trunk monitor created by another application.
dtmGetBriTrunkStatus	Synchronous	Retrieves complete status and statistics for a digital BRI trunk.
dtmGetTrunkStatus	Synchronous	Retrieves complete status and statistics for a digital T1/E1 trunk.
dtmRefreshTrunkStatus	Asynchronous	Requests a status event from a board.
dtmResetCounters	Asynchronous	Resets error accumulators.
dtmSendAlarm	Synchronous	Starts or stops sending an alarm on a digital trunk.
dtmStartTrunkMonitor	Asynchronous	Starts monitoring a specified trunk.
dtmStopTrunkMonitor	Asynchronous	Stops monitoring a specified trunk.

Using the function reference

A prototype of each function is shown with the function description and details of all arguments and return values. A typical function description includes the following information:

Prototype	The prototype lists the function's arguments and includes the following data types: • WORD (16-bit unsigned) • DWORD (32-bit unsigned) • INT16 (16-bit signed) • INT32 (32-bit signed) • BYTE (8-bit unsigned) If a function argument is a data structure, the complete data structure is defined.
Return values	The return value for a function is either SUCCESS or an error code. For asynchronous functions, a return value of SUCCESS indicates the function was initiated; subsequent events indicate the status of the operation.
Events	If events are listed, the function is asynchronous and is complete when the DONE event is returned. If there are no events listed, the function is synchronous. Any additional information such as reason codes and return values is provided in the value field of the event. Refer to Events for details about DTM events.
Example	Example functions that start with Demo are excerpts taken from demonstration function libraries shipped with the product. Example functions that start with my are excerpts taken from sample application programs shipped with the product. The notation `/* ... */` indicates additional code that is not shown.

dtmAttachTrunkMonitor

Enables an application to attach to a trunk monitor service object created by another application when **dtmStartTrunkMonitor** is invoked.

Prototype

DWORD **dtmAttachTrunkMonitor** (CTAHD ***ctahd***, unsigned ***board***, unsigned ***trunk***, DTMHD ****dtmhd***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
board	Board number.
trunk	Specified trunk on the board. The first trunk is 0.
dtmhd	Pointer to the returned trunk monitor object handle.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	Specified ***board*** or ***trunk*** is invalid or is not currently being monitored.
CTAERR_INVALID_HANDLE	Specified ***ctahd*** is invalid.
CTAERR_SVR_COMM	Communication error in server environment.

Details

dtmAttachTrunkMonitor enables an application to attach to a trunk monitor object created by another application. A DTM handle, ***dtmhd***, is returned to the application calling this function. With this handle, the application can access the attached trunk monitor object.

The ***dtmhd*** is valid until the application receives DTM_MONITOR_DONE. Any applications sharing the trunk monitor can call **dtmStopTrunkMonitor** to stop monitoring. The corresponding DONE event is sent to all applications sharing the context. Upon receiving a DONE event with the same handle in the objHd field, the ***dtmhd*** becomes invalid.

dtmGetBriTrunkStatus

Retrieves complete status and statistics for a digital BRI trunk.

Prototype

DWORD **dtmGetBriTrunkStatus** (DTMHD ***dtmhd***, DTM_BRI_TRUNK_STATUS ***status***, unsigned ***size***)

Argument	Description
dtmhd	Handle to the trunk monitor object.

status	Pointer to the DTM_BRI_TRUNK_STATUS structure:
	`typedef struct`
	`{`
	`  DWORD size;  /* size of this structure */`
	`  DWORD board;  /* board number */`
	`  DWORD trunk;  /* trunk number */`
	`  DWORD starttime;  /* when counts started (time_t)*/`
	`  WORD state;  /* state DTM_BRI_STATE_XX */`
	`  WORD type;  /* mode TE or NT */`
	`  DWORD slips;  /* slips accumulator */`
	`  DWORD errors;  /* errors accumulator */`
	`  DWORD receives;  /* receives accumulator */`
	`  DWORD transmits;  /* transmits accumulator */`
	`  WORD b_channel1;  /* B Channel 1 */`
	`  WORD b_channel2;  /* B Channel 2 */`
	`} DTM_BRI_TRUNK_STATUS ;`
	Refer to the Details section for more information on these fields.

size	Size, in bytes, of the DTM_BRI_TRUNK_STATUS structure.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	***dtmhd*** is not a valid monitor handle.
CTAERR_SVR_COMM	Communication error in server environment.

Details

dtmGetBriTrunkStatus retrieves the digital trunk status from the DTM service. The DTM service caches the most recent status and error counters received from the board. **dtmGetBriTrunkStatus** retrieves the status and counters and stores them in a DTM_BRI_TRUNK_STATUS structure.

This is a synchronous function that copies data from the service context into the caller's buffer. The first DWORD contains the size of the returned structure.

The status structure that is stored in the service context is updated each time an event is received from the board. Use **dtmRefreshTrunkStatus** to force an event to be generated.

The physical layer of the NMS ISDN protocol stack must be initialized (with **isdnStartProtocol**) before using the **dtmGetBriTrunkStatus** function. Otherwise the data may not reflect a valid board status (in this case, the state field is equal to DTM_BRI_STATE_NO_USED). For more information, refer to the *NMS ISDN Messaging API Developer's Reference Manual.*

Other information specific to BRI trunks is stored in the type field and determines how the BRI trunk is used. The parameter value (DTM_BRI_TYPE_TE or DTM_BRI_TYPE_NT) describes whether the trunk is used in terminal (TE) or network (NT) mode. The value of the state field is also affected.

The following table lists the state field values for terminal (TE) mode:

Value	Definition
DTM_BRI_STATE_F1	Inactive
DTM_BRI_STATE_F2	Sensing
DTM_BRI_STATE_F3	Deactivated
DTM_BRI_STATE_F4	Awaiting signal
DTM_BRI_STATE_F5	Identifying input
DTM_BRI_STATE_F6	Synchronized
DTM_BRI_STATE_F7	Activated
DTM_BRI_STATE_F8	Lost framing

The following table lists the state field values for network (NT) mode:

Value	Definition
DTM_BRI_STATE_G1	Deactivated
DTM_BRI_STATE_G2	Pending activation
DTM_BRI_STATE_G3	Activated
DTM_BRI_STATE_G4	Pending deactivation

Note: For more details about trunk state values, refer to the ITU-T recommendation I.430.

dtmGetTrunkStatus

Retrieves complete status and statistics for a digital T1/E1 trunk.

Prototype

DWORD **dtmGetTrunkStatus** (DTMHD ***dtmhd***, DTM_TRUNK_STATUS ***status***, unsigned ***size***)

Argument	Description
dtmhd	Handle to the trunk monitor object.

status	Pointer to the DTM_TRUNK_STATUS structure:
	`typedef struct`
	`{`
	`  DWORD size;  /* size of this structure */`
	`  DWORD board;  /* board number */`
	`  DWORD trunk;  /* trunk number */`
	`  DTM_TRUNK_STATE state;  /* alarms and sync status */`
	`  DWORD starttime;  /* when counts started (time_t) */`
	`  DWORD slips;  /* slips accumulator */`
	`  DWORD lineerrors;  /* line code violations (BPVs) */`
	`  DWORD frameerrors;  /* frame bit errors+ CRC errors */`
	`  DWORD es;  /* errored seconds accumulator */`
	`  DWORD ses;  /* severely errored seconds */`
	`  DWORD uas;  /* unavailable seconds */`
	`} DTM_TRUNK_STATUS;`

size	Size, in bytes, of the DTM_TRUNK_STATUS structure.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	***dtmhd*** is not a valid monitor handle.
CTAERR_SVR_COMM	Communication error in server environment.

Details

dtmGetTrunkStatus retrieves the digital trunk status from the DTM service. The DTM service caches the most recent status and error counters received from the board. **dtmGetTrunkStatus** retrieves the status and counters and stores them in a DTM_TRUNK_STATUS structure.

This is a synchronous function that copies data from the service context into the caller's buffer.

The same status structure is used for both E1 and T1 boards. The first DWORD contains the size of the returned structure.

The status structure that is stored in the service context is updated each time an event is received from the board. If full reporting is not enabled (as specified in the reportmask field in DTM_START_PARMS), the data may not reflect the current state of the board unless **dtmGetTrunkStatus** was immediately preceded by an event. Use **dtmRefreshTrunkStatus** to force an event to be generated.

See also

dtmStartTrunkMonitor

dtmRefreshTrunkStatus

Requests a status event from the board.

Prototype

DWORD **dtmRefreshTrunkStatus** (DTMHD ***dtmhd***)

Argument	Description
dtmhd	Handle to the trunk monitor object.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	***dtmhd*** is not a valid monitor handle.
CTAERR_SVR_COMM	Communication error in server environment.

Events

Event	Description
DTMEVN_TRUNK_STATUS	The value field contains a 4-byte DTM_TRUNK_STATE structure. When the trunk is operational, all bytes are set to 0. The objHd field contains the ***dtmhd***.

Details

dtmRefreshTrunkStatus sends a request to the board to report the current alarm status. The board sends a trunk status event immediately upon receiving this request. Use **dtmGetTrunkStatus** to retrieve the updated trunk status.

Note: dtmSendAlarm is not used with BRI trunks.

dtmResetCounters

Resets error accumulators.

Prototype

DWORD **dtmResetCounters** (DTMHD ***dtmhd***)

Argument	Description
dtmhd	Handle to the trunk monitor object.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	***dtmhd*** is not a valid monitor handle.
CTAERR_SVR_COMM	Communication error in server environment.

Events

Event	Description
DTMEVN_TRUNK_STATUS	The value field contains a 4-byte DTM_TRUNK_STATE structure. When the trunk is operational, all bytes are set to 0. The objHd field contains the ***dtmhd***.

Details

dtmResetCounters sends a request to the board to reset its accumulators and start time. This request forces a status event to be sent to the host from the board.

dtmSendAlarm

Starts or stops sending an alarm on a digital trunk.

Note: dtmSendAlarm cannot be used with BRI trunks.

Prototype

DWORD **dtmSendAlarm** (DTMHD ***dtmhd***, unsigned ***alarm***)

Argument	Description
dtmhd	Handle to the trunk monitor object.
alarm	Alarm to send. Use DTM_SEND_NO_ALARMS to disable. Refer to the Details section for alarm settings.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	***dtmhd*** is not a valid monitor handle.
CTAERR_SVR_COMM	Communication error in server environment.

Details

dtmSendAlarm starts or stops sending a specified alarm on a digital trunk to the remote end of the trunk. Alarms from the board to the application running on the host are not affected by this operation.

alarm can be one of the following settings:

Alarm setting	Value	Description
DTM_SEND_NO_ALARMS	0	Auto (board generates yellow alarms when appropriate)
DTM_SEND_AIS	1	AIS (all ones signal)
DTM_SEND_LOF	2	Loss of frame (yellow)
DTM_SEND_TS16AIS	3	TS16AIS (E1 only)

dtmStartTrunkMonitor

Starts monitoring a specified trunk.

Prototype

DWORD **dtmStartTrunkMonitor** (CTAHD ***ctahd***, unsigned ***board***, unsigned ***trunk***, DTMHD ****dtmhd***, DTM_START_PARMS ****parms***)

Argument	Description
ctahd	Handle returned by **ctaCreateContext** or **ctaAttachContext**.
board	Board number.
trunk	Specific trunk on the board. The first trunk is 0.
dtmhd	Pointer to the returned monitor handle.

parms	Pointer to a parameter structure. Set this to NULL to use default values. The DTM_START_PARMS structure is:
	`typedef struct`
	`{`
	`  DWORD size;`
	`  DWORD maxevents;`
	`  DWORD reportmask;`
	
	`} DTM_START_PARMS;`
	Refer to the Details section for a description of these fields.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	Specified trunk number is invalid.
CTAERR_FUNCTION_ACTIVE	Already monitoring the specified trunk in the current context.
CTAERR_SVR_COMM	Communication error in server environment.

Events

Event	Description
DTMEVN_MONITOR_DONE	Monitoring ended. The size field contains a board number in the high word and a trunk number in the low word. The value field of the event contains an error code. The objHd field of this event contains the ***dtmhd***.
DTMEVN_MONITOR_STARTED	Monitoring started. The value, size, and objHd fields are the same as for DTMEVN_TRUNK_STATUS.
DTMEVN_TRUNK_STATUS	The value field contains a 4-byte DTM_TRUNK_STATE structure. When the trunk is operational, all bytes are set to 0. The objHd field contains the ***dtmhd***.

Details

dtmStartTrunkMonitor starts monitoring a particular digital trunk and returns a DTM handle, ***dtmhd***. Monitoring is active until it is stopped with **dtmStopTrunkMonitor**, until the DTM service is closed, or until the context associated with ***ctahd*** is destroyed. The ***dtmhd*** is valid until the application receives the DTMEVN_MONITOR_DONE event from the call to **dtmStopTrunkMonitor**.

dtmStartTrunkMonitor can be called multiple times on the same context for different trunks. The same trunk can be simultaneously monitored by multiple contexts.

If the context is being shared by another application, the application receives DTMEVN_MONITOR_STARTED. When this event is received, the other application automatically attaches to the trunk monitor service object. The objhd field in DTMEVN_MONITOR_STARTED contains a unique handle that the application can use.

Note: For more information on shared service objects, refer to the *Natural Access Service Developer's Reference Manual*.

parms points to a DTM_START_PARMS parameter structure. If ***parms*** is NULL, the default parameter values are used. Use NMS parameter management functions to change default parameter values for a processing context.

The DTM_START_PARMS structure includes the following fields:

- maxevents defines the maximum number of status events that the DTM service can generate in a one-second interval. Its default setting is 1. If status changes occur after the event limit is reached, there will be an event at the beginning of the next one-second interval. Events that are solely the result of changes in the errored seconds counters are not included in the event limit.
- reportmask controls when the DTM services sends events. Its default setting is 1. It can be set to any of the following values:

Value	Sends events...
0	Only in response to **dtmRefreshTrunkStatus**.
1	Only when going in or out of alarm (ignoring type).
2	On any change in received alarms.
4	On any status or synchronization change (including loss of signal).
8	On any slip.
0x10	On change in errored seconds counter.
0x20	On change in failed seconds counter.
0x40	On change in severely errored seconds counter.

In an application that wants to know when a trunk fails and when it is restored, use the default value of 1 for reportmask. Set reportmask to 0x7F to get events when anything changes.

DTMEVN_MONITOR_STARTED containing the current alarm status is always sent as soon as the monitoring is started.

dtmStartTrunkMonitor fails if the board type does not support monitoring.

dtmStopTrunkMonitor

Stops monitoring a specified trunk.

Prototype

DWORD **dtmStopTrunkMonitor** (DTMHD ***dtmhd***)

Argument	Description
dtmhd	Handle to the trunk monitor object.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	***dtmhd*** is not a valid monitor handle.
CTAERR_SVR_COMM	Communication error in server environment.

Events

Event	Description
DTMEVN_MONITOR_DONE	Monitoring ended. The size field contains a board number in the high word and a trunk number in the low word. The objHd field of this event contains the ***dtmhd***. The value field of the event contains either an error code or the following reason: CTA_REASON_STOPPED Monitoring was stopped by **dtmStopTrunkMonitor**.

Details

dtmStopTrunkMonitor terminates monitoring of the board and trunk implied by the specified ***dtmhd.*** After DTMEVN_MONITOR_DONE is received, the ***dtmhd*** is no longer valid.

See also

dtmStartTrunkMonitor

6 Data structures

Data structures overview

The DTM service includes the following data structures:

Structure	Description
DTM_BRI_TRUNK_STATUS	Stores the status and error statistics for BRI trunks.
DTM_TRUNK_STATE	Describes the current alarm and synchronization state of a trunk. This structure is included in the DTM_TRUNK_STATUS structure.
DTM_TRUNK_STATUS	Stores trunk information, alarms, trunk status, and error statistics.
DTM_START_PARMS	Determines the maximum number of events to send per second and when to send the events. This structure is referenced in the call to **dtmStartTrunkMonitor**.

DTM_BRI_TRUNK_STATUS

DTM_BRI_TRUNK_STATUS contains current information about trunk status and error statistics for BRI trunks. This data can be obtained through an asynchronous function call (**dtmGetBriTrunkStatus**). The structure is defined as follows:

```
typedef struct
{
    DWORD size;                 /* size of this structure      */
    DWORD board;                /* board number                */
    DWORD trunk;                /* trunk number                */
    DWORD starttime;            /* when counts started (time_t)*/
    WORD state;                 /* state DTM_BRI_STATE_XX      */
    WORD type;                  /* mode TE or NT               */
    DWORD slips;                /* slips accumulator           */
    DWORD errors;               /* errors accumulator          */
    DWORD receives;             /* receives accumulator        */
    DWORD transmits;            /* transmits accumulator       */
    WORD b_channel1;            /* B Channel 1                 */
    WORD b_channel2;            /* B Channel 2                 */
} DTM_BRI_TRUNK_STATUS;
```

DTM_TRUNK_STATE

DTM_TRUNK_STATE is a 4-byte data structure that describes the current alarm and synchronization state of a trunk. This structure is included in DTM_TRUNK_STATUS and in the status event. The structure is defined as follows:

```
typedef struct
{
     BYTE alarms;                       /*     0 = no alarm
                                         *     1 = Far end loss of frame ("yellow")
                                         *     2 = Alarm Indication Signal ("blue")
                                         *     4 = Loss of frame (T1 "red") E1: Loss of
                                         *          sync, excessive BER
                                         *     8 = (E1) Far end loss of MF sync
                                         *     0x10 = (E1) TS16AIS
                                         */

     BYTE sync;                         /*     0 = in sync
                                         *     1 = loss of signal
                                         *     2 = no (frame) sync
                                         *     4 = no multiframe sync
                                         *     8 = No CRC Multiframe Sync
                                         */

     BYTE alarmsent;                    /*     0 = no alarms
                                         *     1 = sending loss of frame
                                         *     2 = sending AIS
                                         *     4 = sending loss of multiframe (E1 CAS)
                                         *     8 = sending TS16AIS (E1)
                                         */
     BYTE  spare;
} DTM_TRUNK_STATE ;
```

DTM_TRUNK_STATUS

DTM_TRUNK_STATUS contains current information about alarms, trunk status, and error statistics. The same structure is used for both T1 and E1 trunks. This data can be obtained by a synchronous function call (**dtmGetTrunkStatus**). The structure is defined as follows:

```
typedef struct
{
    DWORD size;                     /* size of this structure      */
    DWORD board;                    /* board number                */
    DWORD trunk;                    /* trunk number                */
    DTM_TRUNK_STATE state;          /* alarms and sync status      */
    DWORD starttime;                /* when counts started (time_t)*/
    DWORD slips;                    /* slips accumulator           */
    DWORD lineerrors;               /* line code violations (BPVs) */
    DWORD frameerrors;              /* frame bit errors+ CRC errors*/
    DWORD es;                       /* errored seconds accumulator */
    DWORD ses;                      /* severely errored seconds    */
    DWORD uas;                      /* unavailable seconds         */
} DTM_TRUNK_STATUS;
```

DTM_START_PARMS

DTM_START_PARMS is referenced in the call to **dtmStartTrunkMonitor**. It determines the maximum number of events to send per second and when to send the events. The structure is defined as follows:

```
typedef struct
{
    DWORD size;               /* size of this structure           */
    DWORD maxevents;          /* max number of events per second  */
    DWORD reportmask;         /* controls when to send events     */
} DTM_START_PARMS ;
```

See **dtmStartTrunkMonitor** for valid values.

7 trunklog demonstration program

trunklog overview

trunklog passively monitors for alarm events from digital trunks. It writes to **stdout** whenever the alarm state of a digital trunk changes, as the following example output shows:

```
15:52:26  Board 1 Trunk 0: In service
15:52:37  Board 1 Trunk 0: Loss of frame
15:52:38  Board 1 Trunk 0: Loss of frame Far end alarm indication
15:52:54  Board 1 Trunk 0: Far end alarm indication
15:52:55  Board 1 Trunk 0: In service
```

Running trunklog

Run *trunklog* with AG and CG boards that have a coprocessor and digital line interfaces. To compile and use the demonstration program, Natural Access must be installed on your system.

Compiling trunklog

To compile *trunklog*, open a command window and enter the following command:

- Under Windows, go to the directory *\nms\ctaccess\demos\trunklog* and enter:

  ```
  nmake
  ```

- Under UNIX, go to the directory */opt/nms/ctaccess/demos/trunklog* and enter:

  ```
  make
  ```

Using trunklog

To run the demonstration program:

1. Set up the configuration file to describe the board and software configuration.

   Refer to the board's installation and developer's manual for more information about configuration files.

2. Boot the board to make configuration file changes effective, and leave it running to monitor errors.

3. Enter the following command:

   ```
   trunklog
   ```

To exit *trunklog*, press **F3** or **ESC**.

Program structure and coding features

This topic explains the demonstration program structure and its routines in detail. Use it as a model when developing your own application.

The main function

The **main** function of *trunklog* starts by calling **initconsole**. **initconsole** sets the keyboard to raw mode if *trunklog* is compiled for UNIX.

main then calls **init_ct_access**. **init_ct_access** performs the following tasks:

Task	Description
1	Calls **ctaSetErrorHandler** to specify the **errorhandler** function as the error handling routine for *trunklog*. If an error occurs, **errorhandler** sends a message to **stderr** and calls **doexit** to terminate the program. **errorhandler** does not report an error when calls to **dtmStartTrunkMonitor** return CTAERR_FUNCTION_NOT_AVAIL or CTAERR_INVALID_BOARD. The **start_monitor** function is not interrupted if it attempts to start up the DTM service on a board number for which there is no corresponding board.
2	Calls **ctaInitialize** to establish a list of available services for the application, including the DTM and ADI services.
3	Calls **ctaCreateQueue** to create an event queue.
4	Calls **ctaCreateContext** to create a context to handle the incoming call.
5	Calls **ctaOpenServices** to start up the DTM and ADI services.
6	Calls **ctaWaitEvent** to wait for CTAEVN_OPEN_SERVICES_DONE with a value field of 0, meaning that all services started successfully. If it receives this event, it returns. If it times out or if value does not equal 0, it displays an error message and calls **doexit** to terminate the program.

main then calls **start_monitor**, which performs the following tasks:

Task	Description
1	Sets the parameters in the DTM_START_PARMS structure. It sets maxevents to 2, so the DTM service will return two events per second. It sets reportmask to DTM_REPORT_ALARMS, so DTM will report any change in the alarm state.
2	Starts monitoring trunks on boards 0 through 15 by calling **dtmStartTrunkMonitor** iteratively for each board number. Even if **dtmStartTrunkMonitor** returns an error (for example, there is something wrong with a given board, or no board exists for the board number), **start_monitor** continues until it has tried all 16 board numbers.

If monitoring services are successfully started on at least one board, **main** then calls **process_events**. **process_events** performs the following tasks:

Task	Description
1	Checks repeatedly to see if you pressed the **ESC** or **F3** key. If you press one of these keys, **process_events** calls **doexit** to terminate the program.
2	Calls **ctaWaitEvent** repeatedly and waits for the following events: DTMEVN_MONITOR_STARTED Monitoring started on a trunk. **process_events** calls **dtmGetTrunkStatus** (or (**dtmGetBriTrunkStatus**) to determine on which board and trunk the monitoring started, and then calls **display_data** to send the trunk's initial conditions to **stdout**. DTMEVN_TRUNK_STATUS An alarm state changed on a trunk. **process_events** calls **dtmGetTrunkStatus** (or **dtmGetBriTrunkStatus)** to determine on which board and trunk the state change took place, and then calls **display_data** to send the change to **stdout**. CTAEVN_WAIT_TIMEOUT **ctaWaitEvent** timed out. **process_events** calls **ctaWaitEvent** again. DTMEVN_MONITOR_DONE Monitoring ended on a board. **process_events** determines on which board monitoring ended, and sends a message to **stdout**. ADIEVN_BOARD_ERROR An unexpected error occurred on a board. **process_events** sends a message to **stderr**.

8 Errors and events

Errors

All Natural Access functions return SUCCESS (0), or an error code indicating that the function failed and a reason for the failure.

There are no specific DTM service error codes because the DTM service functions use Natural Access error codes.

Natural Access error codes are defined in the include files *ctaerr.h*, *swidef.h*, and *vcedef.h*. The error codes are prefixed with CTAERR, SWIERR, or VCEERR, respectively. Error codes can also appear in the value field of a DONE event. Use the CTA_IS_ERROR macro to determine if a value is an error.

Refer to the *Natural Access Developer's Reference Manual* for a description of the Natural Access errors.

Events

All events in the Natural Access environment are represented by a C data structure, as shown in the following generic CTA_EVENT:

```
typedef struct
{
    DWORD id;            /* event ID (LIBEVN_xxx in 'libdef.h')                */
    CTAHD ctahd;         /* CTA context handle                                 */
    DWORD timestamp;     /* timestamp                                          */
    DWORD userid;        /* use-supplied ID                                    */
    DWORD size;          /* size of buffer if buffer is not NULL               */
                         /* otherwise, may contain event-specific data         */
    void *buffer;        /* buffer pointer                                     */
    DWORD value;         /* Event status or event-specific data                */
    DWORD objHd;         /* object handle                                      */
}   CTA_EVENT;
```

This structure, returned by **ctaWaitEvent**, informs the application which event occurred on which context, and provides additional information specific to the event. The LIB prefix in LIBEVN relates the event to a specific NMS library of functions. For example, the CTA prefix indicates Natural Access and DTM indicates the DTM service.

The event structure contains the following fields:

Field	Description
id	Contains an event code defined in the library header file. All DTM events are prefixed with DTMEVN_ (for example, DTMEVN_SOMETHING_HAPPENED).
ctahd	Contains the context handle (returned from **ctaCreateContext**).
timestamp	Contains the time an event was created in milliseconds since midnight. The resolution for AG board events is 10 milliseconds.
userid	Contains the user-supplied ID. This field is unaltered by Natural Access and facilitates asynchronous programming. Its purpose is to correlate a context with an application object and context when events occur.
size	Specifies the size (bytes) of the area pointed to by buffer. If the buffer is NULL, this field can hold an event-specific value. Can contain the ***dtmhd.***
buffer	Points to data returned with the event. The field contains an application process address and the event's size field contains the actual size of the buffer.
value	Contains a reason code or an error code. This is an event-specific value.
objHd	Contains the ***dtmhd*** in DTM events.

The following events are specific to the DTM service:

Event	Hexadecimal	Decimal	Description
DTMEVN_MONITOR_DONE	0XC2101	794881	Monitoring ended.
DTMEVN_MONITOR_STARTED	0XC2001	794625	Monitoring started on a specified trunk.
DTMEVN_TRUNK_STATUS	0XC2002	794626	Trunk status information is available.

Index