<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>9</td>
</tr>
<tr>
<td>1.1. About the IPTGate Demo</td>
<td>9</td>
</tr>
<tr>
<td>1.2. The IPTGate Hardware Components</td>
<td>9</td>
</tr>
<tr>
<td>1.3. What Does the IPTGate Demo Do?</td>
<td>10</td>
</tr>
<tr>
<td>1.4. How Does the IPTGate Demo Work?</td>
<td>11</td>
</tr>
<tr>
<td>1.4.1. Handling a PSTN Call</td>
<td>11</td>
</tr>
<tr>
<td>1.4.2. Handling an IP Call</td>
<td>12</td>
</tr>
<tr>
<td>1.4.3. Disconnecting the Calls</td>
<td>12</td>
</tr>
<tr>
<td>1.5. Where To Go From Here</td>
<td>12</td>
</tr>
<tr>
<td>2. Running the Demo</td>
<td>15</td>
</tr>
<tr>
<td>2.1. Introduction</td>
<td>15</td>
</tr>
<tr>
<td>2.2. How Does the IPTGate Demo Choose a Channel?</td>
<td>15</td>
</tr>
<tr>
<td>2.3. Downloading the IPTGate Firmware</td>
<td>15</td>
</tr>
<tr>
<td>2.4. Modifying the IPTGate.cfg File</td>
<td>16</td>
</tr>
<tr>
<td>2.5. Running the IPTGate Demo</td>
<td>19</td>
</tr>
<tr>
<td>2.5.1. Starting the Demo</td>
<td>19</td>
</tr>
<tr>
<td>2.5.2. Answering a PSTN Incoming Call</td>
<td>21</td>
</tr>
<tr>
<td>2.5.3. Answering an IP Incoming Call</td>
<td>22</td>
</tr>
<tr>
<td>2.5.4. Disconnecting a Call</td>
<td>22</td>
</tr>
<tr>
<td>3. Application Flow</td>
<td>23</td>
</tr>
<tr>
<td>3.1. General</td>
<td>23</td>
</tr>
<tr>
<td>3.2. Programming Model</td>
<td>25</td>
</tr>
<tr>
<td>3.3. Initialization</td>
<td>25</td>
</tr>
<tr>
<td>3.3.1. Demo Related Initialization</td>
<td>25</td>
</tr>
<tr>
<td>3.3.2. IPLink Related Initialization</td>
<td>26</td>
</tr>
<tr>
<td>3.3.3. PSTN Card Related Initialization</td>
<td>27</td>
</tr>
<tr>
<td>3.4. Retrieving Events</td>
<td>28</td>
</tr>
<tr>
<td>3.4.1. Retrieving D/xxx (SRL) Events</td>
<td>28</td>
</tr>
<tr>
<td>3.4.2. Retrieving IPT Events</td>
<td>29</td>
</tr>
<tr>
<td>3.5. Using State Machines</td>
<td>29</td>
</tr>
<tr>
<td>3.6. PSTN Inbound Call</td>
<td>29</td>
</tr>
<tr>
<td>3.7. IP Inbound Call</td>
<td>31</td>
</tr>
<tr>
<td>4. PSTN Inbound Call</td>
<td>33</td>
</tr>
<tr>
<td>4.1. Wait_For_Call State</td>
<td>33</td>
</tr>
</tbody>
</table>
4.1.1. Receiving the PSTN Call ................................................................. 33
4.1.2. Routing Between DM3 and PSTN Boards ........................................ 34
4.1.3. Making an IP Outbound Call .......................................................... 34
4.2. Wait_For_Connect State ................................................................. 35
4.3. Wait_For_Disconnect (Disconnect Supervision) ..................................... 36
   4.3.1. PSTN Disconnect ........................................................................... 37
   4.3.2. IP Disconnect .............................................................................. 37
4.4. Wait_For_Idle State(Retrieving Call Status) ......................................... 38
   4.4.1. Get Call Info ................................................................................ 38
4.5. Wait_For_Release (Exiting Call) ........................................................ 39
5. IP Inbound Call .................................................................................. 41
   5.1. Wait_For_Call State ......................................................................... 41
       5.1.1. Receiving the IP Call ............................................................... 42
   5.2. Wait_For_Connect State ................................................................. 43
       5.2.1. Making a PSTN Outbound Call ................................................ 44
   5.3. Wait_For_Disconnect State (Disconnect Supervision) .......................... 44
       5.3.1. IP Disconnect ........................................................................... 45
   5.4. Wait_For_Idle State (Retrieving Call Status) ....................................... 45
   5.5. Wait_For_Release (Exiting the Call) ................................................... 46
6. Advanced Topics .................................................................................. 49
   6.1. Collecting Routing Information From the PSTN Call ............................ 49
   6.2. Connecting the IP call to the PSTN Call ............................................. 49
   6.3. Billing ............................................................................................. 49
   6.4. Least Cost Routing .......................................................................... 50
   6.5. PSTN Disconnect Supervision ......................................................... 50
Appendix A - The IPTGate.cfg file .......................................................... 51
Appendix B - Make Call Log File .............................................................. 55
Appendix C - Call Offering Log File .......................................................... 59
Appendix D - Digital State Diagrams ......................................................... 63
Appendix E - Application Foundation Code Reference ............................... 65
   What is the Application Foundation Code? ............................................. 65
   Foundation Code Architecture ............................................................... 66
      Core Foundation Code and Core Services .......................................... 67
      Product-Specific Layers and Services ................................................. 67
      Standard and Miscellaneous Services ............................................... 67
   Component Interface Attributes ........................................................... 68
Internal Data Structures ................................................................. 69
Core Foundation Code Modules .......................................................... 70
Dm3CompProcIoCompletion( ) Process a message completion ............. 70
Dm3CompEnableSyncMode( ) Set the host component for synchronous mode .......................................................................................... 72
Dm3CompSetAsyncParams( ) Set the asynchronous parameters for the host component ......................................................................... 73
Dm3CompEnableAsyncMode( ) Set the host component for asynchronous mode ....................................................................................... 74
Dm3CompRecvMsg( ) Prepare the host component to receive an asynchronous message ................................................................. 75
Standard messages Sends various standard messages to the DM3 components ....................................................................................... 76
IPLink Resource Layer Functions .......................................................... 79
Dm3Tsp Data Structure Definition ....................................................... 79
Dm3TscInit( ) Initializes a TSC instance ............................................ 81
Dm3TscCleanUp ( ) Releases the resources and handles owned by a TSC instance .............................................................................. 84
TSC Command Functions ..................................................................... 85
Dm3TscMakeCall( ) Places an outgoing call ......................................... 85
Dm3TscAnswerCall( ) Seizes the line of an incoming call ..................... 88
Dm3TscAnswerCall( ) Seizes the line of an incoming call ..................... 90
Dm3TscAcceptCall( ) Accepts an incoming call ................................... 92
Dm3TscDropCall( ) Drops a call ......................................................... 94
Dm3TscReleaseCall( ) Releases the call ID .......................................... 96
Dm3TscRejectCall( ) Rejects an incoming call ..................................... 98
Dm3TscGetChanState( ) Retrieves channel state information .......... 100
Error Codes ......................................................................................... 101
Dm3TscGetCallState( ) Gets the current call state .............................. 102
Dm3TscEvtHndlr( ) Main, top level handler for all Tsc events .............. 104
Event Handling .................................................................................. 106
OnTscMakeCallCmplt( ) Event Handler for TSC_MakeCallCmplt message ......................................................................................... 106
OnTscStdMsgEvtDetected( ) Handler for STD_MsgEvtDetected events on a Tsc instance ............................................................ 108
OnTscCallInfoEvent( ) Handler for Call Information events on a Tsc instance ..................................................................................... 110
OnTscChanStateEvent( ) Handler for Channel State transition events on a Tsc instance .............................................................. 112
Limitations ................................ ................................ ................................ 113
Error Codes ................................ ................................ ............................... 113
OnTscCallStateEvent( ) Handler for Call State transition events on a Tsc
instance. ................................ ................................ ........................... 114
NetTSC Functions ................................ ................................ .......................... 116
Dm3NetTscInit ................................ ................................ ......................... 116
NetTSCResetSession ................................ ................................ ................. 117
Dm3NTscNonStdCmd ................................ ................................ .............. 118
Dm3NTscUII ................................ ................................ ............................ 119
NETTSCClusterInit ................................ ................................ .................. 120
NETTSCClusterGetAllComps ................................ ................................ ... 121
NETTSCClusterListen ................................ ................................ .............. 122
NETTSCClusterUnlisten ................................ ................................ ........... 123
NETTSCClusterGetXmitSlot ................................ ................................ .... 124
NETTSCClusterRelease ................................ ................................ ............ 125
NETTSCClusterCleanup ................................ ................................ ........... 126
NETTSCClusterGetComponent ................................ ................................ 127
NETTSCClusterGotComponent ................................ ................................ 128
Index ...................................................................................................... 129
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IPTGate Demo Components</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Typical Topology</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>Running the IPTGate Demo</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>Incoming IP Call</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>State Diagram - Analog PSTN Inbound Call</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>State Diagram - Analog IP Inbound Call</td>
<td>31</td>
</tr>
<tr>
<td>7</td>
<td>PSTN Inbound: Wait For Call</td>
<td>33</td>
</tr>
<tr>
<td>8</td>
<td>PSTN Inbound: Wait for Connect</td>
<td>36</td>
</tr>
<tr>
<td>9</td>
<td>PSTN Inbound: Wait for Disconnect</td>
<td>37</td>
</tr>
<tr>
<td>10</td>
<td>PSTN Inbound: Wait for Idle</td>
<td>38</td>
</tr>
<tr>
<td>11</td>
<td>PSTN Inbound: Wait for Release</td>
<td>39</td>
</tr>
<tr>
<td>12</td>
<td>IP Inbound: Wait for Call</td>
<td>41</td>
</tr>
<tr>
<td>13</td>
<td>IP Inbound: Wait for Connect</td>
<td>43</td>
</tr>
<tr>
<td>14</td>
<td>IP Inbound: Wait for Disconnect</td>
<td>45</td>
</tr>
<tr>
<td>15</td>
<td>IP Inbound: Wait for Idle</td>
<td>46</td>
</tr>
<tr>
<td>16</td>
<td>IP Inbound: Wait for Release</td>
<td>47</td>
</tr>
<tr>
<td>17</td>
<td>Digital PSTN Inbound Call</td>
<td>63</td>
</tr>
<tr>
<td>18</td>
<td>Digital IP Inbound Call</td>
<td>64</td>
</tr>
<tr>
<td>19</td>
<td>DM3 Direct Interface Foundation Code Architecture</td>
<td>66</td>
</tr>
</tbody>
</table>
1. Introduction

This chapter describes the IPTGate demo and suggests various ways to use the demo to learn about creating an IP Telephony application.

1.1. About the IPTGate Demo

The IPTGate demo is a host based application that demonstrates using the Dialogic’s DM3 IPLink platform, together with a Dialogic SCbus PSTN interface card to build a PSTN – IP gateway. The demo can be used as sample code for those who want to start developing their application with a working application. The demo uses many of the functions provided by the IPLink platform. However, it is not designed to implement a complete gateway. In particular it lacks features such least cost routing, etc. Where possible, this document tries to point out where things can be done differently, or where features may be added. However, the variety of options is large, and one demo cannot cover all the different possibilities.

1.2. The IPTGate Hardware Components

The IPTGate demo requires the following hardware components:

- Pentium 200MHz or better
- At least 32 Mbytes of memory
- Windows NT 4.0
- A Dialogic analog or digital SCbus compatible PSTN board, e.g. D/160SC-LS, D/240SC-T1, D/300SC-E1
- SCbus cable and H.100 adapter
- A DM3 IPLink board:
  - DM/IPLink-T1 or DM/IPLink-E1
    - Requires NIC (Network Interface Card) that connects to an IP network
  - DM/IPLink-T1_NIC or DM/IPLink-E1_NIC
    - Network Interface is built-in
- Additional 85 MB of available hard disk space
IPTGate Demo
(IP - PSTN Gateway)
User’s Guide

- CD-ROM drive
- VGA or higher-resolution display adapter
- Microsoft Mouse or compatible pointing device

In addition you will must connect a PSTN line (from a PABX or a line simulator) to the D/xxx board, and an IP network cable (typically Ethernet) to the NIC or Network Interface on the IPLink board. And lastly — a telephone to call into the demo. Figure 1 shows the components of the IPTGate demo, using a separate NIC. If you are using a DM/IPLink-T1_NIC or DM/IPLink-E1_NIC board, connect the IP network cable directly to the IPLink board.

![Figure 1. IPTGate Demo Components](image)

1.3. What Does the IPTGate Demo Do?

The IPTGate demo allows you to connect to gateways on an IP network, and establish calls from telephone to telephone via the IP network. It also allows you to connect H.323 terminals on the IP network and connect a call from the terminal to a telephone via one of the gateways. Figure 2 shows a typical topology for demonstrating the capabilities of the IPTGate demo. Note that the two PABXs that are shown can be a single PABX. Also note that more than one PSTN line can be connected to a single IPTGate demo gateway.
1. How Does the IPTGate Demo Work?

The IPTGate demo can receive calls from either the PSTN or the IP. The IPTGate demo uses a configuration file (`IPTGate.cfg`) to determine parameters that are associated with a particular call. The configuration file allows you to configure different channels with different properties. The exact description of the `IPTGate.cfg` file, as well as a description of the different configuration properties, are described later in this guide.

1.4.1. Handling a PSTN Call

A call that arrives from the PSTN needs to be routed to either a destination PSTN number (via another gateway) or to an H.323 terminal. The IPTGate demo uses the `IPTGate.cfg` file to determine the destination IP address as well as the (optional) destination PSTN number (Remote Phone Number). The IPTGate demo initiates an IP (H.323) call to the destination IP address. If the configuration file indicates a PSTN destination number then that number is passed to the destination gateway during the call establishment procedure.
Once the destination gateway has answered the H.323 call, the IPTGate demo connects the PSTN call to the IP call (via the SCBus). An audio path is now established between the PSTN call and the destination IP station.

### 1.4.2. Handling an IP Call

A call that arrives from the IP network needs to be routed to a PSTN number. That number may arrive as part of the call establishment procedure (if the call was originated by another IPTGate for example). If the destination number had arrived during call establishment, then the IPTGate demo uses that number to call on the PSTN. If no destination number was included in the call establishment procedure, then the IPTGate demo uses the `IPTGate.cfg` file to determine the destination number to call (Local Phone Number). Once the IPTGate demo answers and connects the call on the IP network, it initiates (dial out) a call on the PSTN and connects the two calls (via the SCBus). This allows the calling party to hear the call progress tones on the local PSTN (see discussion later for other possibilities on when to connect the calls).

### 1.4.3. Disconnecting the Calls

The IPTGate demo monitors both the PSTN call and the IP call for a disconnect indication. While the IP call provides a certain indication (i.e. an H.323 drop indication) the IPTGate demo may not recognize a disconnect on the PSTN side, since it only monitors a loop current drop on the PSTN.

### 1.5. Where To Go From Here

The IPTGate demo can be used in a variety of ways:

- Run the demo
  - Download the DM3 and Dialogic firmware
  - Modify the `IPTGate.cfg` file
  - Run the demo
- Explore the IPTGate code
  - Read this manual (what’s in it)
  - Look in the source code
1. Introduction

- Modify the code
- Compile the code
- Build your own gateway
  - Read the rest of the documentation
  - Design the state machines
  - Re-use the code from the demo
2. Running the Demo

2.1. Introduction

NOTE: This document assumes that you have installed the necessary HW and SW on the demo machine and are ready to download and run the demo.

This chapter presents the main flow for running the demo. It explains what you can expect to see on the screen, as well as what you can expect to happen in the system.

2.2. How Does the IPTGate Demo Choose a Channel?

When a call comes from the PSTN, the call is connected to a particular telephone line. This telephone line is associated with a channel number by its physical connection. For example, if it is connected to the first connection on a D/160LS, then the call is handled by channel #1.

When a call arrives from the IP network, there is no direct association of a channel, since there are no physical connections. The call is answered by a NetTSP cluster (an “IPT channel”. See the IPLink User’s Guide for more information on NetTSP clusters). The IPTGate demo uses the built in resource manager of the DM3 driver to select an available NetTSP cluster to handle the call. It connects the IP call to the highest available PSTN channel.

2.3. Downloading the IPTGate Firmware

Follow the directions for installing and configuring the IPLink SDK in Installing and Configuring the IPLink SDK for Windows NT. The following parameters must be set as follows for the IPTGate demo to work properly:

- SCbus Clock Master

  The DM3 board must provide the clocking. Configure the D/xxx Board as SCbusClockMaster = NONE.
**IPTGate Demo**  
*(IP - PSTN Gateway)*  
**User’s Guide**

- **PCD Files**
  Identify a PCD file that matches your configuration. The PCD files supplied with the release are described in the Release Catalog.

- **Set the SRAM Out Time to 12**

### 2.4. Modifying the *IPTGate.cfg* File

Before running the demo, modify the *IPTGate.cfg* file to reflect your system environment. Click on the Configuration File icon to open the file, or open the file from `samples\ipt\iptgate\debug\IPTGate.cfg`. At the minimum, change the PSTN telephone numbers that are associated with the telephones and PABX that you are using.

The configuration file includes a prioritized coder list. Coder0 is the preferred coder. This list is passed to the *TSC_MsgMakeCall* structure or *TSC_MsgAnswerCall* structure. **Note:** The *TSC_MsgMakeCall* structure (and *TSC_MsgAnswerCall* structure) reflects the coder prioritization by the order of the coders in the data structure. The highest priority coder is listed first. “DON’T_CARE” is a valid coder option, however it must appear last in the coder list. Otherwise, coders listed following the “DON’T_CARE” will be ignored.

The H.323 component checks the coder capability of the remote side during the call creation. If the preferred coder is not supported, it will check the next coder in the list. If no coder compatibility is found, the connection fails. In that case one of the following events is sent to the application:

- As a response to Make Call: *TSCEvtCallState_Type_Failed*
- As a response to Call Offering: *TSCEvtCallState_Type_Idle*

The application must then release the call to free the channel.

Below is an example of the *IPTGate.cfg* file. The fields that you may want to change include the source and destination IP address, the local and remote phone numbers, as well as information about the coder that should be used when originating a call toward the IP network.
2. Running the Demo

Source = 146.152.187.51
Channel = 1
{
   Destination = 146.152.187.51
   RemotePhoneNumber = 36
   LocalPhoneNumber = 28
   Coder0
   { Type = g711MuLaw
     FramesPerPkt = 1
     FrameSize = 30
     Rate = 0
     VAD = 0
   }
   Display = IPTGate_Chan1
   UUI = User_to_User_1
   UII = 255
   NonStdCmd = NSC_Chan1
}
IPTGate Demo  
(IP - PSTN Gateway)  
User’s Guide

Channel = 2
{
  Destination = 146.152.187.51  
  RemotePhoneNumber = 30  
  LocalPhoneNumber = 28  
  Coder0
  {
    Type = g711MuLaw  
    FramesPerPkt = 1  
    FrameSize = 30  
    Rate = 0  
    VAD = 0  
  }
  Coder1
  {
    Type = g723  
    FramesPerPkt = 1  
    Rate = 0  
    VAD = 0  
  }
  Coder2
  {
    Type = Don't_Care  
    FramesPerPkt = Don't_Care  
    FrameSize = Don't_Care  
    Rate = Don't_Care  
    VAD = Don't_Care  
  }
  Display = IPTGate_Chan2  
  UUI = User_to_User_2  
  UII = 255  
  NonStdCmd = NSC_Chan2
}

Channel = 3
{
  Destination = 146.152.187.51  
  RemotePhoneNumber = 28  
  LocalPhoneNumber = 29  
  Coder0
  {
    Type = g711MuLaw  
    FramesPerPkt = 1  
    FrameSize = 30  
    Rate = 0  
    VAD = 0  
  }
  Display = IPTGate_Chan3  
  UUI = User_to_User_3  
  UII = 255  
  NonStdCmd = NSC_Chan3
}

Channel = 4-6
{
  Destination = 146.152.187.51  
  RemotePhoneNumber = 2019933255  
  LocalPhoneNumber = 27  
  Coder0
}
2. Running the Demo

```{Type = g711MuLaw
FramesPerPkt = 1
FrameSize = 30
Rate = 0
VAD = 0
}
Display = IPTGate_Chan4
UI1 = User_to_User_4
UII = 255
NonStdCmd = NSC_Chan4
}
```

2.5. Running the IPTGate Demo

2.5.1. Starting the Demo

Click on “IPTGate” from the IPT SDK menu to run the IPTGate demo using the default settings. The demo may also be launched from the Windows NT command line, using the following switches:

<table>
<thead>
<tr>
<th>Switch</th>
<th>Action</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>-f</td>
<td>Identifies the front end:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = analog</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 = digital T1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 = digital E1</td>
<td></td>
</tr>
<tr>
<td>-b</td>
<td>Identifies the board number</td>
<td>0</td>
</tr>
<tr>
<td>-r</td>
<td>Sets the number of rings before answering the call on the PSTN</td>
<td>2</td>
</tr>
<tr>
<td>-n</td>
<td>Sets the number of channels</td>
<td>The lesser of PSTN lines or NetTSC clusters</td>
</tr>
<tr>
<td>-c &lt;filename&gt;</td>
<td>Configuration file name</td>
<td>IPTGate.cfg</td>
</tr>
</tbody>
</table>

Switch | Action | Default
---|---|---
-e | Encoding type: 
m = µLaw 
a = A-law | m

NOTE: The default encoding type is µLaw. If you are using A-law encoding, perform the following additional two steps:

1. Verify that the “PCM Encoding” parameter for the PSTN board matches the configuration.

2. Open the Dialogic Configuration Manager and double-click on the PSTN board name.

   Select the “Dialogic Bus” tab and click on the PCMEncoding parameter.

   Select either ALAW or MULAW from the pull-down menu in the Value field in the Edit window.

Each channel is initialized and the log is displayed in a DOS window. See Appendix B and Appendix C for the log file from a typical “MakeCall” and a “ReceiveCall” session.
2. Running the Demo

2.5.2. Answering a PSTN Incoming Call

A PSTN call arrives when one calls on a PSTN line to the IPTGate demo. You can follow the call trace, which is displayed in a DOS window. See Appendix B and Appendix C for the log file from a “MakeCall” and a “ReceiveCall” session.
IPTGate Demo
(IP - PSTN Gateway)
User's Guide

Figure 4. Incoming IP Call

2.5.3. Answering an IP Incoming Call

An IP call arrives when another H.323 station calls the IPTGate demo. This can be either another gateway, or an H.323 terminal.

2.5.4. Disconnecting a Call

The IPTGate demo monitors both the IP side and the PSTN side for disconnect supervision.

NOTE: For the sake of simplicity, on the PSTN side the IPTGate demo monitors only loop current drops. When you disconnect a telephone call, your PABX may not generate a loop current drop toward the PSTN side. In such case the IPTGate demo will not see the disconnect of the PSTN call, and will not drop the call toward the IP network. In such a case unplug the PSTN line from the D/xxx board momentarily, to generate a loop current drop.
3. Application Flow

3.1. General

This section presents a closer examination of the IPTGate demo, by looking into its structure and source code. The IPTGate demo source code is included in the following files:

- **IPTGATE.CFG** - The demo configuration file
- **DM3TSC.C** - DM3 convenience functions
- **DM3CLUST.C** - DM3 cluster convenience functions
- **DM3COMP.C** - DM3 component convenience functions
- **DM3NTSC.C** - NetTSC component convenience functions
- **NTSCCLST.C** - NetTSC cluster convenience functions
- **DM3TSCH.C** - TSC component application handler
- **DM3CLSTH.C** - Cluster application handler
- **DM3NTSCH.C** - NetTSC component application handler
- **NTSCCLSH.C** - NetTSC cluster application handler
- **GATEDBG.C** - Debugging & error functions
- **GATEMAIN.C** - Main file (including MAIN loop)
- **GATEPARS.C** - The demo configuration file parsing functions
IPTGate Demo
(IP - PSTN Gateway)
User’s Guide

GATEPSTN.C - PSTN-specific functions
GATESTAT.C - State machine functions
GATEDEFS.H - Gateway definitions
GATEVARS.H - Global variables
GATESTRCH.H - Demo structure (including MAIN Structure Session)
GATEDBG.H - Debugging & error definitions and macros for GATEDBG.C
DM3CLSTH.H - Definitions & macros for DM3CLSTH.C
DM3CLUST.H - Definitions & macros for DM3CLUST.C
DM3COMP.H - Definitions & macros for DM3COMP.H
DM3NTSC.H - Definitions & macros for DM3NTSC.C
DM3NTSCH.H - Definitions & macros for DM3NTSCH.C
DM3TSC.H - Definitions & macros for DM3TSC.C
DM3TSCH.H - Definitions & macros for DM3TSCH.C
MAIN.H - Function prototype for MAIN.C
NTSCCLSH.H - Definitions & macros for NTSCCLSH.C
NTSCCLST.H - Definitions & macros for
3. Application Flow

NTSCCLST.C

PSTNFP.H - Function prototype for GATEPSTN.C

STATFP.H - Function prototype for GATESTAT.C

PARSFP.H - Function prototype for GATEPARS.C

IPTGATE.MAK - Visual C++ make file

IPTGATE.MDP - Visual C++ project file

3.2. Programming Model

The IPTGate demo is designed to operate in single thread mode. Channels are designed to operate as independent state machines. The synchronization between SRL events (the D/xxx events) and IPT specific events is done by using the WinNT I/O Completion Port mechanism. Because of the nature of a single threaded application, the state machines do not, and should not, block the operation on any other operation but the wait for the I/O completion port.

NOTE: The application programming framework also allows multi-thread operation. It is not demonstrated in the IPTGate demo.

3.3. Initialization

3.3.1. Demo Related Initialization

1. Get any arguments from the command line option.
2. Create I/O Completion Port (WinNT API).
3. Find the DEMO max available channels (the smaller of available PSTN channels or available NetTSC clusters).
4. Reset the demo data structure. Initialize all channels’ state to GATE_WAIT_DETECT_CMPLT.
5. Read information from the configuration file (IPTGate.cfg), and update data structure accordingly.
6. Create waitForKey thread to receive keyboard input. 
   Input is valid in the Connected state only. The following keyboard input is recognized:
   - “U” or “u”  
     Send User Input Indication (UII)
   - “N” or “n” 
     Send NonStandard Command
   - ^c or “Q” or “q”  
     Clean-up and quit application

3.3.2. IPLink Related Initialization

The IPLink initialization procedure described in this section is defined in the clusterInit() and NetTSCCompInit() functions in the GATEMAIN.C file.

1. clusterInit()
   - Call NETTSCClusterInit()
     - Opens Mercury Message Path device by calling mntEnumMpathDevice() and CreateFile()
     - Gets the Mpath address by calling mntGetMpathAddr()
     - Enable ExitNotify for the Mpath device
     - Indicate the NetTSC cluster to be used
     - Allocates NetTSC cluster and finds the SCbus component descriptors in the same cluster
   - Call NETTSCClusterGetAllComps()
     - Get the NetTSC component instance descriptor for the previously allocated cluster.

2. NetTSCCompInit()
   - Get the NetTSC transmit timeslot
3. Application Flow

- Call **Dm3NetTSCInit()**
  - Initialize NetTSC structure
  - Opens Mercury Message Path device by calling `mntEnumMpathDevice()` and `CreateFile()`
  - Gets the Mpath address by calling `mntGetMpathAddr()`
  - Enable Exit Notify for the Mpath device
  - Indicate the NetTSC component (found in `clusterGetAllComps`) instance to be used
- Call **Dm3CompSetAsyncParams()**
  - Attach DM3_IPT_KEY to I/O Completion Port created in the Demo global initialization
- Call **Dm3CompEnableAsyncMode()**
  - Set the async field to TRUE
- Call **IPTEnableAllEvts()**
  - Enable list of events relevant to the NetTSC component instance.
  Once the Std_MsgDetectXEvtsCmplt is received by the application, the channel state transitions to WAIT_FOR_CALL.

3.3.3. PSTN Card Related Initialization

The IPTGate demo can be run with either a digital or analog PSTN board. The description detailed in this section assumes an analog PSTN board. See Appendix D for the state diagrams related to using a digital PSTN board.

The PSTN initialization procedure described in this section is defined in the `pstnOpenFrontEnd()` function in the `GATEPSTN.C` file.

For every demo channel (variable `gateChannel`), the application:

1. Opens a voice device, by calling `dx_open()`.
2. Sets the voice channels to ON HOOK, by calling `dx_sethook()`.
IPTGate Demo
(IP - PSTN Gateway)
User's Guide

3. Enables the detection of a Call Status Transition event, by calling
   `dx_setevtmsk()` . The events detected are: DM_RINGS and DM_LCOFF.
4. Sets the number of rings (transferred from command line) to wait before
detecting DM_RINGS, by `dx_setrings()`.
5. Removes all user-defined tones, by `dx_deltones()`.
6. Defines a simple dual frequency cadence tone for disconnect detection, by
calling `dx_blddtcad()`.
7. Enables detection of user-defined tones on the channel, by calling
   `dx_addtone()`.
8. Get `xmitSlot`.

3.4. Retrieving Events

The IPTGate demo uses the WinNT I/O Completion Port mechanism to retrieve
events.

1. Create an I/O Completion Port without associating it with a file.
2. Attach SRL_KEY and DM3_IPT_KEY to the I/O Completion Port and
   associate the I/O Completion Port with the asynchronous file created (the
   asynchronous device handle for retrieving messages).
   • DM3_IPT_KEY is used to receive events from the DM3 card
   • SRL_KEY is used to receive events from the PSTN card

3.4.1. Retrieving D/xxx (SRL) Events

1. Use the SRL event data retrieval functions:
   • `sr_getevtttype()` — gets event type for the current event
   • `sr_getevtdatap(0)` — returns a pointer to the variable data associated
     with the current event
   • `sr_getevtddev(0)` — gets the Dialogic handle for the current event
2. Call the related state machine function according to the data retrieved.
3. Application Flow

3.4.2. Retrieving IPT Events

1. Use the `GetQueuedCompletionStatus()` function to get the `OVERLAPPED` structure.
2. Call `Dm3CompProcIoCompletion()` to retrieve the event.
3. Call the appropriate call back function to handle the event and to call the appropriate state machine function.

3.5. Using State Machines

There is an endless loop (while(1)) in the `main()` function in the `Gatemain.c` file. In that loop the application waits for an event by calling the `GetQueuedCompletionStatus()` function (WinNT API).

Initialize all channels to the WAIT_FOR_CALL state.

As soon as an event is received, the event type, the channel number and the reason (reason for the event, if there is one) are analyzed, and the new state machine function is called.

After all the operations are performed within the channel’s event state, the next state machine function is updated according to the event received.

The following state diagrams describe the call-states for inbound calls from the PSTN to the demo (gateway), and inbound calls from the IP to the demo (gateway). Each state is represented by an ellipse containing the state name, the event(s) associated with that state, and the actions performed by the application. The states are connected with arrows indicating the valid state changes.

3.6. PSTN Inbound Call

Figure 5 describes the call states and the state transitions when the demo (gateway) receives a call from the PSTN.
IPTGate Demo
(IP - PSTN Gateway)
User’s Guide

Figure 5. State Diagram - Analog PSTN Inbound Call
3. Application Flow

3.7. IP Inbound Call

Figure 6 describes the call states and the state transitions when the demo (gateway) receives a call from the IP.

Figure 6. State Diagram - Analog IP Inbound Call
4. PSTN Inbound Call

This chapter describes the procedure and state transitions for connecting an inbound call from the PSTN to the IP network.

4.1. Wait_For_Call State

The application waits for a call event in the WAIT_FOR_CALL state.

4.1.1. Receiving the PSTN Call

The application receives the event **DE_RINGS** from the PSTN. It routes the call between the PSTN board and the DM3 board and sets the PSTN line to Off-hook. It then sends a **TSC_MsgMakeCall** message to the NetTSC component instance associated with the call. The state transitions to WAIT_FOR_CONNECT.

![Figure 7. PSTN Inbound: Wait_For_Call](image-url)
4.1.2. Routing Between DM3 and PSTN Boards

Routing: Listen & Assign/Activate (DM3 listen to PSTN)

```c
NETTSCClusterListen( /*pointer to the NetTSC Cluster*/
    &Session[channel].NetTscClust),
    /*The PSTN timeslot to listen to*/
    (USHORT)timeslot);
```

4.1.3. Making an IP Outbound Call

The application sends `TSC_MsgMakeCall` using the data contained in the configuration file.

```c
/* Set array of information to make call */
for(infoCount = 0; infoCount < (CFGParm.maxTxCoders);
    infoCount++) {
    KV_Info[infoCount].unKeyId = TSC_KVSet_Key_CallInfo;
    KV_Info[infoCount].unId = CallInfo_TxCoder;
    KV_Info[infoCount].unLength = sizeof(NetTSC_Coder_t);
    KV_Info[infoCount].lpData = &(void)(CFGParm.TxCoder[infoCount]);
}
```

```c
KV_Info[infoCount].unKeyId = TSC_KVSet_Key_CallInfo;
KV_Info[infoCount].unId = CallInfo_Display;
KV_Info[infoCount].unLength = strlen(CFGParm.display)+1;
KV_Info[infoCount].lpData = (void *)CFGParm.display;
infoCount++;
```

```c
KV_Info[infoCount].unKeyId = TSC_KVSet_Key_CallInfo;
KV_Info[infoCount].unId = CallInfo_PhoneList;
KV_Info[infoCount].unLength = strlen(CFGParm.phoneList)+1;
KV_Info[infoCount].lpData = (void *)CFGParm.phoneList;
infoCount++;
```

```c
KV_Info[infoCount].unKeyId = TSC_KVSet_Key_CallInfo;
KV_Info[infoCount].unId = CallInfo_UUI;
KV_Info[infoCount].unLength = strlen(CFGParm.UUI)+1;
KV_Info[infoCount].lpData = (void *)CFGParm.UUI;
infoCount++;
```

```c
KV_Info[infoCount].unKeyId = TSC_KVSet_Key_NULL;
KV_Info[infoCount].unLength = 0;
infoCount++;
```
4. PSTN Inbound Call

/* Set source address of call */
strcpy(srcAddr,"TA:");
strcat(srcAddr,CFGParm.srcAddr);
strcat(srcAddr,":1720");

/* Set destination address of call: i.e. destination a computer on other end start with the basic TSC make call structure */
strcpy(destAddr,"TA:");
strcat(destAddr,CFGParm.destAddr);
strcat(destAddr,":1720");  /* Add Port id */

Dm3TscMakeCall(lpTsc, /* pointer to DM3TSC structure */
destAddr, /* destination IP address */
srcAddr, /* source IP address */
FALSE, /* set for call progress */
KV_Info, /* array of KV-Set */
infoCount); /* number of KV-Set elements */

4.2. Wait_For_Connect State

The application receives a `TSC_EvtCallState_Type_Connected` event from the NetTSC component instance when the far gateway or terminal answers the call. The call is now connected and the state transitions to WAIT_FOR_DISCONNECT.

If, for any reason, the call should fail before it is connected, the application receives a `TSC_EvtCallState_Type_Failed` message from the NetTSC component instance, or a `DE_LCOFF` and `DE_TONEON` from the PSTN. The call state transitions to WAIT_FOR_IDLE.
4.3. **Wait For Disconnect (Disconnect Supervision)**

The application can receive three types of messages while a call is connected:

- Disconnect message
- Non-standard message
- User Input Indication message

When a call is disconnected, the application receives either a `TSC_MsgMakeCallCmplt` event from the NetTSC component instance, or `DE_LCOFF` and `DE_TONEON` from the PSTN. It sets the PSTN line to On-hook, un-routes the call and issues a `TSC_MsgDropCall` message to the NetTSC component instance. The call state transitions to `WAIT_FOR_IDLE`.

The application can also receive two non-standard events:
4. PSTN Inbound Call

- **NetTSC_H245Data_Type_NonStdCmd**
- **NetTSC_H245Data_Type_UserInputIndication**

When the application receives either of these events, it prints the received data on the screen and remains in the WAIT_FOR_DISCONNECT state.

![Diagram](Image)

Figure 9. PSTN Inbound: Wait for Disconnect

4.3.1. PSTN Disconnect

Unrouting: Unlisten & Deactivate/Unassign (DM3 unlisten to PSTN)

```c
NETTSCClusterUnlisten( /*pointer to the NetTSC cluster*/
  &{Session[channel].NetTscClust});
```

4.3.2. IP Disconnect

The application sends **TSC_MsgDropCall** to the NetTSC component instance.
Dm3TscDropCall(lpTsc, /* pointer to DM3TSC structure */
    /* reason for dropping call */
    CallStateR_RemoteTermination);

4.4. Wait_For_Idle State (Retrieving Call Status)

The application receives a \textit{TSC\_EvtCallState\_Type\_Idle} event and then gets call information (e.g., duration time, RTCP info) by sending a \textit{TSC\_MsgGetCallInfo} message to the TSC component instance.

Once the application receives the requested call information, it sends a \textit{TSC\_MsgReleaseCall} message and transitions to the WAIT\_FOR\_RELEASE state.

\textbf{Figure 10. PSTN Inbound: Wait for Idle}

4.4.1. Get Call Info

/* Set array for get call info message */
4. PSTN Inbound Call

InfoArr[0] = CallInfo_RTCPInfo; /*RTCP Information*/
InfoArr[1] = CallInfo_CallDurationTime; /*Call Duration Time*/

Dm3TscGetCallInfo(lpTsc, /* pointer to DM3TSC structure */
                   2, /* number for call information elements */
                   InfoArr); /* the information elements */

4.5. Wait_For_Release (Exiting Call)

The application receives a TSC_EvtCallState_Type_Null event and resets all
data structures to their default values. The call-state transitions to
WAIT_FOR_CALL.

![Diagram](image)

Figure 11. PSTN Inbound: Wait for Release
5. IP Inbound Call

This chapter describes the procedure and state transitions for connecting an inbound call from the IP network to the PSTN.

5.1. Wait_For_Call State

The application waits for a call event in the WAIT_FOR_CALL state.

The application receives the event **TSC_EvtCallState_Type_Offered** from the IP. It sends **TSC_MsgGetCallInfo** and a **TSC_MsgAnswerCall** messages to the NetTSC component instance associated with the call. The state transitions to WAIT_FOR_CONNECT.

---

**Figure 12. IP Inbound: Wait for Call**
5.1.1. Receiving the IP Call

Getting Call Info

The application sends a `TSC_MsgGetCallInfo` message to the NetTSC component instance to get the receiving coder type and the PSTN number to dial.

```c
/* Set array for get call info message */
InfoArr[0] = CallInfo_PhoneList; /* Local extention to call */
InfoArr[1] = CallInfo_CallerId;  /* Who’s calling – Computer IP */
InfoArr[2] = CallInfo_Display;   /* Who’s calling – Caller name */
InfoArr[3] = CallInfo_UUI;      /* User to User Information */

Dm3TscGetCallInfo(lpTsc, /* pointer to DM3TSC structure */
  4,  /* number for call information elements */
  InfoArr);  /* the information elements */
```

Answering the IP Call

The application sends a `TSC_MsgAnswerCall` message to the NetTSC component instance. The data sent is taken from the configuration file.

```c
/* Prepare the KVSet for answering the call */
for(infoCount = 0; infoCount < (CFGParm.maxTxCoders);
    infoCount++) {
    KV_Info[infoCount].unKeyId = TSC_KVSet_Key_CallInfo;
    KV_Info[infoCount].unId = CallInfo_TxCoder;
    KV_Info[infoCount].unLength = sizeof(NetTSC_Coder_t);
    KV_Info[infoCount].lpData = s((void)(CFGParm.TxCoder[infoCount]));
}

KV_Info[infoCount].unKeyId = TSC_KVSet_Key_NULL;
KV_Info[infoCount].unLength = 0;
infoCount++;

rBool = Dm3TscAnswerCall(lpTsc, /* pointer to DM3TSC structure */
  2,  /* number of rings */
  KV_Info,  /* KV-Set array */
```
5. **IP Inbound Call**

5.2. **Wait_For_Connect State**

The application receives a `TSCEvtCallState_Type_Connected` event from the NetTSC component instance. It then sets the PSTN Off-hook, dials the call to the PSTN and routes the call over the SCbus. The call is now connected and the state transitions to WAIT_FOR_DISCONNECT.

If, for any reason, the call should fail before it is connected, the application receives a `TSCEvtCallState_Type_Failed` message from the NetTSC component instance. The call state transitions to WAIT_FOR_IDLE.

![Figure 13. IP Inbound: Wait for Connect](image-url)
5.2.1. Making a PSTN Outbound Call

The application dials the local phone number according to the information it received from the TSC_MsgGetCallInfo message. If no information was received, it is taken from the configuration file.

5.3. Wait_For_Disconnect State (Disconnect Supervision)

The application can receive three types of messages while a call is connected:

- Disconnect message
- Non-standard message
- User Input Indication message

When a call is disconnected, the application receives either a TSC_EvtCallState_Type_Disconnect event from the NetTSC component instance, or DE_LCOFF and DE_TONEON from the PSTN. It sets the PSTN line to On-hook, un-routes the call and issues a TSC_MsgDropCall message to the NetTSC component instance. The call state transitions to WAIT_FOR_IDLE.

The application can also receive two non-standard events:

- NetTSC_H245Data_Type_NonStdCmd
- NetTSC_H245Data_Type_UserInputIndication

When the application receives either of these events, it prints the received data on the screen and remains in the WAIT_FOR_DISCONNECT state.
5. IP Inbound Call

5.3.1. IP Disconnect

The application sends \textit{TSC\_MsgDropCall} to the NetTSC component instance.

5.4. Wait\_For\_Idle State (Retrieving Call Status)

The application receives a \textit{TSC\_EvtCallState\_Type\_Idle} event and then gets call information (e.g., duration time, RTCP info) by sending a \textit{TSC\_MsgGetCallInfo} message to the TSC component instance.

Once the application receives the requested call information, it sends a \textit{TSC\_MsgReleaseCall} message and transitions to the \textit{WAIT\_FOR\_RELEASE} state.

Figure 14. IP Inbound: Wait for Disconnect
Figure 15. IP Inbound: Wait for Idle

5.5. Wait_For_Release (Exiting the Call)

The application receives a TSCEvtCallState_Type_Null event and resets all data structures to their default values. The call-state transitions to WAIT_FOR_CALL.
5. IP Inbound Call

Figure 16. IP Inbound: Wait for Release
6. Advanced Topics

The IPTGate demo is a basic demo that shows simple usage of the DM3 IPLink platform. Some enhancements that may be useful include:

6.1. Collecting Routing Information From the PSTN Call

There are many ways that a gateway may collect the destination address, both IP address as well as a target PSTN number. This may include:

- Using DTMF signals
- Using ANI features
- Using a sophisticated IVR system
- Using a sophisticated directory service.

6.2. Connecting the IP call to the PSTN Call

The IPTGate demo connects an originating IP call to the PSTN call as soon as it completes dialing. This allows the calling party to hear the call progress tones. However, this indicates to the calling gateway (or terminal) that the call is established. In some applications, you may want to hold off the connection (answering the IP call) until after it determines the status of the PSTN call (by using the call progress analysis features of the PSTN card). It is possible that in this case the IPTGate demo will play some “music on hold” or other messages to the calling party.

6.3. Billing

The IPTGate demo includes a billing part. Several options for billing include:
Start billing as soon as the remote IP station had answered the call. This has the following problems:

- If the remote station answers the call immediately after the PSTN dial, then the billing may start before the call is established. Moreover, the call may not complete successfully, while the billing is activated.

- If the remote station answers the call only after it determines a positive connection on the PSTN call, then the calling party will not hear the call progress tones.

- Have the gateway answer the IP call immediately after it dials out on the PSTN to allow for call progress tones. Use the Dialogic non-standard channel that is established between gateways to exchange billing information (such as the result of the PSTN call).

### 6.4. Least Cost Routing

The IPTGate demo does not demonstrate least cost routing.

### 6.5. PSTN Disconnect Supervision

The IPTGate demo uses a very simple method for determining PSTN disconnect. More sophisticated methods may be used.
Appendix A
The IPTGate.cfg file

Source = 146.152.187.51
Channel = 1
{
    Destination = 146.152.187.51
    RemotePhoneNumber = 36
    LocalPhoneNumber = 28
    Coder0
    {
        Type = g711MuLaw
        FramesPerPkt = 1
        FrameSize = 30
        Rate = 0
    }
}
IPTGate Demo
(IP - PSTN Gateway)
User's Guide

VAD = 0
)
Display = IPTGate_Chan1
UUI = User_to_User_1
UII = 255
NonStdCmd = NSC_Chan1
}

Channel = 2
{
Destination = 146.152.187.51
RemotePhoneNumber = 30
LocalPhoneNumber = 28
Coder0
{
Type = g711MuLaw
FramesPerPkt = 1
FrameSize = 30
Rate = 0
VAD = 0
}
Coder1
{
Type = g723
FramesPerPkt = 1
Rate = 0
VAD = 0
}
Coder2
{
Type = Don't_Care
FramesPerPkt = Don't_Care
FrameSize = Don't_Care
Rate = Don't_Care
VAD = Don't_Care
}
Display = IPTGate_Chan2
UUI = User_to_User_2
UII = 255
NonStdCmd = NSC_Chan2
}

Channel = 3
{
Destination = 146.152.187.51
RemotePhoneNumber = 28
LocalPhoneNumber = 29
Coder0
{
Type = g711MuLaw
FramesPerPkt = 1
FrameSize = 30
Rate = 0
VAD = 0
}
Display = IPTGate_Chan3
UUI = User_to_User_3
UII = 255
NonStdCmd = NSC_Chan3

52
Appendix A

Channel = 4-6

Destination = 146.152.187.51
RemotePhoneNumber = 201993255
LocalPhoneNumber = 27
Coder0

  Type = g711µLaw
  FramesPerPkt = 1
  FrameSize = 30
  Rate = 0
  VAD = 0

Display = IPTGate_Chan4
UUI = User_to_User_4
UII = 255
NonStdCmd = NSC_Chan4

}
Appendix B

Make Call Log File

---
ITPGate --- Voice over IP Gateway Demo Program.
Version 1.00 Release 1.00
Copyright (c) 1997, Dialogic Corp.
---

Can only support 3 channels.

TRACE: File: GATEMISC.C Line: 281
Session 1: Destination 146.152.187.74
  PhoneList 28
  LocalPhoneNumber 27
  Coder 0
  FramesPerPacket 3
  FrameSize 30
  Rate 1
  VAD 0
  Display IPTGate_Chan1
  UUI User_to_User_1
  UII 1##
  NonStdCmd NSC_Chan1

TRACE: File: GATEMISC.C Line: 281
Session 2: Destination 146.152.187.55
  PhoneList 29
  LocalPhoneNumber 28
  Coder 2
  FramesPerPacket 1
  FrameSize 30
  Rate 1
  VAD 1
  Display IPTGate_Chan2
  UUI User_to_User_2
  UII 22###
  NonStdCmd NSC_Chan2

TRACE: File: GATEMISC.C Line: 281
Session 3: Destination 146.152.187.51
  PhoneList 28
  LocalPhoneNumber 29
IPTGate Demo
(IP - PSTN Gateway)
User's Guide

Coder 2
FramesPerPacket 1
FrameSize 30
Rate 1
VAD 1
Display IPTGate_Chan3
UII User_to_User_3
UII 333###***
NonStdCmd NSC_Chan3

Waiting for key:
'Q' - to quit
'N' - for Non standard command (Active only in connected state)
'U' - for User Input Indication (Active only in connected state)

WAITING FOR EVENT: INCOMING CALL OR OUTGOING CALL.

TRACE: File: GATESTAT.C Line: 76
In WAIT_FOR_CALL on channel 1
got event DE_RINGS (0x1)

TRACE: File: GATESTAT.C Line: 182
After ROUTE_ALL.

TRACE: File: GATESTAT.C Line: 273
In WAIT_FOR_CONNECT on channel 1
got event TSC_EvtCallState_Type_Connected (0x1232)

TRACE: File: GATESTAT.C Line: 394
In WAIT_FOR_DISCONNECT on channel 1
got event DE_TONEON (0x11)

TRACE: File: GATESTAT.C Line: 478
In WAIT_FOR_IDLE on channel 1
got event TSC_EvtCallState_Type_Idle (0x1239)
Got RTCPInfo LocalSR_TxPackets 474
LocalRR_FructionLost 0
RemoteSR_TxPackets 107
RemoteRR_FructionLost 0
on channel 1
Got 7 second duration time on channel 1

TRACE: File: GATESTAT.C Line: 552
In WAIT_FOR_RELEASE on channel 1
   got Event TSC_EvtCallState_Type_Null (0x1230)
Appendix C
Call Offering Log File

-------------------------------------------------------------------------
IPTGate --- Voice over IP Gateway Demo Program.
Version 1.00 Release 1.00
Copyright (c) 1997, Dialogic Corp.
-------------------------------------------------------------------------
Can only support 3 channels.

TRACE: File: GATEMISC.C Line: 281
Session 1: Destination 146.152.187.74
  PhoneList 28
  LocalPhoneNumber 27
  Coder 0
  FramesPerPacket 3
  FrameSize 30
  Rate 1
  VAD 0
  Display IPTGate_Chan1
  UUI User_to_User_1
  UII 1##
  NonStdCmd NSC_Chann1

TRACE: File: GATEMISC.C Line: 281
Session 2: Destination 146.152.187.55
  PhoneList 29
  LocalPhoneNumber 28
  Coder 2
  FramesPerPacket 1
  FrameSize 30
  Rate 1
  VAD 1
  Display IPTGate_Chan2
  UUI User_to_User_2
  UII 22##**
  NonStdCmd NSC_Chann2

TRACE: File: GATEMISC.C Line: 281
Session 3: Destination 146.152.187.51
  PhoneList 28
IPTGate Demo
(IP - PSTN Gateway)
User's Guide

LocalPhoneNumber 29
Coder 2
FramesPerPacket 1
FrameSize 30
Rate 1
VAD 1
Display IPTGate_Chan3
UUI User_to_User_3
UII 333###***
NonStdCmd NSC_Chan3

Waiting for key:
'O' - to quit
'N' - for Non standard command (Active only in connected state)
'U' - for User Input Indication (Active only in connected state)

WAITING FOR EVENT: INCOMING CALL OR OUTGOING CALL.

TRACE: File: GATESTAT.C Line: 76
In WAIT_FOR_CALL on channel 3
  got event TSC_EvtCallState_Type_Offered (0x123b)
Got CallerId TA:146.152.187.74:1171,NAME:ami givati on channel 3
Got display ami givati on channel 3

TRACE: File: GATESTAT.C Line: 138
Answering call on channel 3

TRACE: File: GATESTAT.C Line: 273
In WAIT_FORCONNECT on channel 3
  got event TSC_EvtCallState_Type_Connected (0x1232)

TRACE: File: GATESTAT.C Line: 291
Dialing (29) on channel 3

TRACE: File: GATESTAT.C Line: 304
ROUTE_ALL after dial on channel 3

TRACE: File: GATESTAT.C Line: 394
In WAIT_FOR_DISCONNECT on channel 3
  got event TSC_EvtCallState_Type_Disconnected (0x1236)

TRACE: File: GATESTAT.C Line: 428
Got Call State Disconnected the reason is 19
Appendix C

TRACE: File: GATESTAT.C Line: 478
In WAIT_FOR_IDLE on channel 3
    got event TSCEvtCallState_Type_Idle (0x1239)
    Got RTCPInfo LocalSR_TxPackets 697
    LocalRR_FructionLost 0
    RemoteSR_TxPackets 107
    RemoteRR_FructionLost 0
    on channel 3
    Got 7 second duration time on channel 3

TRACE: File: GATESTAT.C Line: 552
In WAIT_FOR_RELEASE on channel 3
    got Event TSCEvtCallState_Type_Null (0x1230)
Appendix D

Digital State Diagrams

Figure 17. Digital PSTN Inbound Call
**IPTGate Demo**  
*(IP - PSTN Gateway)*  
**User's Guide**

---

**Figure 18. Digital IP Inbound Call**

- **Wait for Call**: 
  - **NetTSC**: TSC_EVT Call State Type Offered
  - Send Message Get Call Info to NetTSC
  - Route

- **Wait for Connect**: 
  - **NetTSC**: TSC_EVT Call State Type Connected
  - Set Off-hook

- **Wait for Disconnect**: 
  - **NetTSC**: TSC_EVT Call State Type Disconnected
  - Send Message Drop Call to NetTSC

- **Wait for Release**: 
  - **NetTSC**: TSC_EVT Call State Type Idle
  - Send Message Release Call to NetTSC

- **Wait for Idle**: 
  - **NetTSC**: TSC_EVT Call State Type Idle
  - Send Message Get Call Info to NetTSC and Send Message Release Call to NetTSC

- **Wait for Release**: 
  - **NetTSC**: TSC_EVT Call State Type Null
  - Set On-hook

- **Wait for Call**: 
  - **NetTSC**: TSC_EVT Call State Type Idle
  - Reset Data Structure

- **PSTN**: DIGITAL_ONHOOK
  - Send Message Release Call to NetTSC

- **PSTN**: DIGITAL_OFFHOOK
  - Send Message Get Call Info to NetTSC

- **PSTN_INIT**: 
  - **NetTSC**: TSC_EVT Call State Type Null
  - Reset Data Structure

- **PSTN_Release**: 
  - **NetTSC**: TSC_EVT Call State Type Null
  - Set On-hook

- **TSC_EVT Call State Type Offered**: 
  - Send Message Rejected Call to NetTSC

- **TSC_EVT Call State Type Idle**: 
  - Send Message Get Call Info to NetTSC and Send Message Release Call to NetTSC

- **TSC_EVT Call State Type Idle**: 
  - Send Message Get Call Info to NetTSC

- **TSC_EVT Call State Type Null**: 
  - Set On-hook

- **TSC_EVT Call State Type Idle**: 
  - Send Message Get Call Info to NetTSC

- **TSC_EVT Call State Type Idle**: 
  - Send Message Get Call Info to NetTSC and Send Message Release Call to NetTSC

- **TSC_EVT Call State Type Idle**: 
  - Send Message Get Call Info to NetTSC

- **TSC_EVT Call State Type Null**: 
  - Set On-hook

---

*Diagram depicting the flow of digital IP inbound call with various states and message exchanges.*
Appendix E
Application Foundation Code Reference

What is the Application Foundation Code?

The Application Foundation Code for the DM3 Direct Interface for Windows NT is a set of software functions provided with the Development Tool Kits for the DM3-based products. The primary goal of the foundation code is to give the DM3 application developers a jumpstart in their application design. C is the implementation choice for the foundation code. The foundation code is based on asynchronous model using I/O Completion Ports as the event notification mechanism.

The foundation code is organized as a collection of header files, source files, and application handlers. The foundation source files provide the basic DM3 functionality simplifying most of the operations involved in communicating with the firmware. The handlers are skeletal source that need to be completed by the application developer to implement the application logic. It is organized as separate empty functions for each and every incoming event the application has to respond to. The application developer fills up the function for incoming events and links it with the other foundation files.
Foundation Code Architecture

Figure 19. DM3 Direct Interface Foundation Code Architecture
Core Foundation Code and Core Services

The DM3 Direct Interface core foundation code provides services that are product-independent. There are three modules in the core foundation code.

- The host component module handles all the messaging aspects of the DM3 platform. This includes sending and receiving messages, and processing messages. The host component module is not aware of any specifics of DM3 messages; it just acts as a transporter. The host component module provides a mechanism to register a callback function for each resource instance. For example, a TSP resource, conforming to the foundation code, would register a callback function when initializing a host component. This callback function will be called by the host component module so that the resource can interpret the messages.

- The cluster module provides SCbus switching functions that are functionally comparable to the R4 switching functions.

- The stream module abstracts streaming-related aspects of the DM3 platform. This module is not used by the IPLink platform.

Product-Specific Layers and Services

Product-specific layers are built on top of the core foundation code. The IPLink specific layer, for example, contains services that are specific to the TSC and the NetTSC resources that are provided as part of the IPLink platform. This product-specific layer makes use of the core foundation code services for messaging and SCbus switching. Each product that conforms to this foundation code has to publish a set of services/interfaces it supports. Using these interfaces, the application can access technology-specific services.

As the resource layers are built on top of the core foundation code, they need to abide by the rules and guidelines set forth by the core foundation code.

Standard and Miscellaneous Services

Sending and receiving standard component interface messages is common to all components in DM3. Applications would have to directly access these services
for sending standard messages. Thus, the standard message service is provided in the host component module.

Dispatching services are needed so that the application code, which contains the main loop for processing events, can call the appropriate foundation code function to process the event.

**Component Interface Attributes**

The following host component interface attributes are contained in DM3COMP data structures.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMpath</td>
<td>Win32 handle for the DM3 message path to be used to communicate with the firmware component</td>
</tr>
<tr>
<td>qcdHostAddress</td>
<td>DM3 component descriptor for the message path</td>
</tr>
<tr>
<td>qcdFWAddress</td>
<td>DM3 component descriptor for the firmware component</td>
</tr>
<tr>
<td>lpUserInfo</td>
<td>Any information that user of the component wants to associate with this component handle. At component init time, the user of the component (like TSC, ToneGen) would pass this pointer as an argument.</td>
</tr>
<tr>
<td>lpfnCallBack</td>
<td>Callback function registered by the user of component. This function will be called by the component as part of processing a message.</td>
</tr>
<tr>
<td>hIOCP</td>
<td>Handle for IO Completion port to be used.</td>
</tr>
<tr>
<td>dwIOCPKey</td>
<td>Key that is used to indicate any message IO completion for this component.</td>
</tr>
</tbody>
</table>
| bSyncMode      | True → Sends messages in synchronous mode  
False → Sends messages in asynchronous mode |
Appendix E

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwTimeOutInSecs</td>
<td>Timeout values in seconds; to be used when sending messages synchronously</td>
</tr>
<tr>
<td>ucExpectedReplyCount</td>
<td>Number of replies expected for the command</td>
</tr>
<tr>
<td>lpCriticalSection</td>
<td>Pointer to the critical section to be used to protect data structure in multi-threaded environment.</td>
</tr>
</tbody>
</table>

Internal Data Structures

Various elements in the DM3MSGOVERLAPPED data structure are described below.

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overlapped</td>
<td>Win 32 Overlapped structure to be used with this message IO. This has to be the first element in the structure so that the foundation code can perform the above mentioned upcast.</td>
</tr>
<tr>
<td>lpMMB</td>
<td>Pointer to the MMB structure that is associated with this IO.</td>
</tr>
<tr>
<td>lpComp</td>
<td>Pointer to the data structure of the component that sent the message</td>
</tr>
</tbody>
</table>
Core Foundation Code Modules

**Dm3CompProcIoCompletion()**  Process a message completion

**Purpose**

Called by the application code when the main loop retrieves an event associated with a message path.

**Function Signature**

```
Name: LPVOID Dm3CompProcIoCompletion(
LPOVERLAPPED lpOverlapped,  BOOL fOk)

Inputs:
LPOVERLAPPED lpOverlapped
• Pointer to the overlapped structure retrieved from the GetQueuedCompletionStatus call.

BOOL fOk
• Return value of GetQueuedCompletionStatus

Returns: Pointer returned by the application handler

Includes: •

Category: •

Mode: Sync
```

**Internal Operation**

This is a foundation code host component function that is invoked by the application code. The event notification (GetQueuedCompletionStatus) code resides in the application code main loop. When the application sends a message asynchronously, an IO completion packet will be queued when the message IO completes. The application code then retrieves an IO message key from the IO completion port. The application code should then call this foundation code function, passing it the pointer to the overlapped structure. By using an
association technique, this foundation code function extracts the handle for the resource that sent this message and the LPMMB used to send the message. After extracting these two members, this function invokes the appropriate callback handler registered for the host component.

**Pseudocode**

1. Upcast lpOverlapped to DM3MSGOVERLAPPED;
   // For a discussion on DM3MSGOVERLAPPED refer to internal data structure section
2. Retrieve the lpMMB and lpComp from the DM3MSGOVERLAPPED;
3. Retrieve the QMsg and QMsgType from the lpMMB reply;
4. Call the pMsgCallBack function with lpComp, CmdMsgType, MsgType and QMsgRef as arguments.
Dm3CompEnableSyncMode( )  Set the host component for synchronous mode

Purpose

Called by the application code when the application code wants to send messages synchronously to the firmware. All the messages from this host component will be sent in synchronous mode until a Dm3CompEnableAsyncMode( ) call is issued.

Function Signature

Name:  DM3STATUS Dm3CompEnableSyncMode( LPDM3COMP lpComp, USHORT usTimeoutInSecs)

Inputs:  LPDM3COMP lpComp  • Pointer to the component interface attribute structure.
         USHORT usTimeoutInSecs  • Timeout to be used.

Returns:  SUCCESS or FAILURE

Includes:

Category:  Sync

Mode:  Sync

Internal Operation

Sets the bSyncMode component interface attribute to TRUE.
Dm3CompSetAsyncParams( ) Set the asynchronous parameters for the host component

Purpose

This function sets various parameters needed to operate the host component in asynchronous mode.

Function Signature

Name: DM3STATUS Dm3CompSetAsyncParams(PDM3COMP lpComp, HANDLE hIOCP, DWORD dwIocpKey)

Inputs:
- LPDM3COMP lpComp • Pointer to the component interface attribute structure
- HANDLE HIOCP • IO Completion Port to be used
- DWORD dwIocpKey • IO Completion Key for this component

Returns: SUCCESS or FAILURE •

Includes: •

Category: •

Mode: Sync

Internal Operation

Sets the bSyncMode component interface attribute to FALSE.
Dm3CompEnableAsyncMode()  Set the host component for asynchronous mode

Purpose

This function is called by the application code when the application code wants to send messages asynchronously to the firmware. All the messages from this host component will be sent in asynchronous mode until a Dm3CompEnableSyncMode() call is issued.

Function Signature

Name:  DM3STATUS Dm3CompEnableAsyncMode(
Inputs:  LPDM3COMP lpComp,  USHORT usTimeoutInSecs)

Returns:  SUCCESS or FAILURE

Includes:

Category:  Sync

Mode:  Sync

Internal Operation

Sets the bSyncMode component interface attribute to FALSE.
Dm3CompRecvMsg() Prepare the host component to receive an asynchronous message

Purpose

This function is called by the application code when the application code wants to receive asynchronous events from a firmware resource.

Function Signature

Name: DM3STATUS Dm3CompRecvMsg( LPDM3COMP lpComp, ULONG ulMsgType)

Inputs:
- LPDM3COMP lpComp • Pointer to the component structure
- ULONG ulMsgType • Asynchronous message type that the application code is interested in receiving

Returns: SUCCESS or FAILURE •

Includes: •

Category: •

Mode: Sync

Internal Operation

Constructs an MMB big enough to hold the asynchronous message and sets the matching criteria as detailed above.
**Standard messages** Sends various standard messages to the DM3 components

**Purpose**

These functions are called by the application code when the application code wants to send a standard component interface message to a firmware component. Most of the functions listed here are self-explanatory.

**Function Signatures**

<table>
<thead>
<tr>
<th>Name:</th>
<th>DM3STATUS  Dm3CompSetParm( LPDM3COMP lpComp, Std_MsgSetParm_Num_t ParmKey, Std_MsgSetParm_Val_t ParmValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs:</td>
<td>•</td>
</tr>
<tr>
<td>Returns:</td>
<td>SUCCESS or FAILURE</td>
</tr>
<tr>
<td>Includes:</td>
<td>•</td>
</tr>
<tr>
<td>Category:</td>
<td>•</td>
</tr>
<tr>
<td>Mode:</td>
<td>•</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>DM3STATUS  Dm3CompSetParmDef( LPDM3COMP IpComp, Std_MsgSetParm_Num_t ParmKey)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs:</td>
<td>•</td>
</tr>
<tr>
<td>Returns:</td>
<td>SUCCESS or FAILURE</td>
</tr>
<tr>
<td>Includes:</td>
<td>•</td>
</tr>
<tr>
<td>Category:</td>
<td>•</td>
</tr>
<tr>
<td>Mode:</td>
<td>•</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>DM3STATUS  Dm3CompSetAllParmsDef( LPDM3COMP IpComp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs:</td>
<td>•</td>
</tr>
<tr>
<td>Returns:</td>
<td>SUCCESS or FAILURE</td>
</tr>
<tr>
<td>Includes:</td>
<td>•</td>
</tr>
<tr>
<td>Category:</td>
<td>•</td>
</tr>
<tr>
<td>Mode:</td>
<td>•</td>
</tr>
</tbody>
</table>
### Name: **DM3STATUS Dm3CompGetParm( LPDM3COMP lpComp,**
\hspace{1cm}Std_MsgSetParm_Num_t ParmKey)\)**

**Inputs:**

- 

**Returns:** SUCCESS or FAILURE

**Includes:**

- 

**Category:**

- 

**Mode:**

---

### Name: **DM3STATUS Dm3CompDetectEvt( LPDM3COMP lpComp,**
\hspace{1cm}Std_MsgDetectEvt_Type_t EvtType,**
\hspace{1cm}Std_MsgDetectEvt_Label_t Label,**
\hspace{1cm}Std_MsgDetectEvt_RetAddr_t qcdRetAddr)\)**

**Inputs:**

- 

**Returns:** SUCCESS or FAILURE

**Includes:**

- 

**Category:**

- 

**Mode:**

---

### Name: **DM3STATUS Dm3CancelEvt( LPDM3COMP lpComp,**
\hspace{1cm}Std_MsgCancelEvt_Type_t MsgType)\)**

**Inputs:**

- 

**Returns:** SUCCESS or FAILURE

**Includes:**

- 

**Category:**

- 

**Mode:**

---

### Name: **DM3STATUS Dm3CompArmRTC( LPDM3COMP lpComp,**
\hspace{1cm}Std_MsgArmRTC_Action_t Action,**
\hspace{1cm}Std_MsgArmRTC_Label_t Label)\)**

**Inputs:**

- 

**Returns:** SUCCESS or FAILURE

**Includes:**

- 

**Category:**

- 

**Mode:**

---
**DM3STATUS Dm3CompDisarmRTC( LPDM3COMP lpComp, Std_MsgDisarmRTC_Label_t Label)**

- **Inputs:**
- **Returns:** SUCCESS or FAILURE
- **Includes:**
- **Category:**
  - **Mode:**

**DM3STATUS Dm3CompTest( LPDM3COMP lpComp)**

- **Inputs:**
- **Returns:** SUCCESS or FAILURE
- **Includes:**
- **Category:**
  - **Mode:**

**Internal Operation**

Builds and sends appropriate message.


IPLink Resource Layer Functions

The foundation code resource layer for the IPLink platform has services for two resources. They are:

- TSC (part of the TSP resource)
- NetTSC (part of the NetTSP resource)

Dm3Tsp Data Structure Definition

This data structure definition extends the base component definition for the TSC specific component.

typedef struct {
    DM3COMP     theComp;    // The common component structure
    LPVOID      lpUserInfo; // A general purpose pointer for app use
    WORD        unTrunkId;  // Which T1 or E1 trunk in the system
    WORD        unBearerChanId; // Timeslot on above trunk.
    WORD        unCallId;   // Id number of any active call or -1
    WORD        nChannelState; // Current channel state
    BOOL        fBusy;     // Whether instance is processing a command
} DM3TSC, *LPDM3TSC

- ComplInfo
  The base component description common to all DM3 components. It is statically declared making it simpler and more flexible for the user to implement and initialize this data whether it is on the heap or stack or statically declared.

- lpUserInfo
  A general purpose pointer for use by the Application. This would be used to link the component instance data structure to a higher level data structure such as might be done to implement an aggregation relationship, etc…

- TrunkId
  The TrunkId associated with this TSC resource.

- BearerChanId
  The timeslot associated with this TSC resource.
**CallId**  
The Call Identification number or “call handle” of any currently active call. This data is managed by the framework. It is initialized during either an “offered” event or a TSC_MsgMakeCallCmplt, and reset during a release call command. If no call is active, this will have a value of 0xFFFF.

**OriginatingId**  
The originating phone number associated with this line.

**ChannelState**  
The channel state is managed by the framework. This field is updated every time a channel state event occurs. These events are always enabled for this component when it is initialized.

**Fbusy**  
A flag maintained by the framework that denotes whether this instance is currently waiting for a reply to a command.
Dm3TscInit( ) Initializes a TSC instance

Purpose

This function will initialize a TSC instance data structure. This initialization allows the events related to the instance to be automatically detected, routed and processed by the application framework.

Function Signature

**Name:** PVOID Dm3TscInit( LPDM3COMPTSC pTspInfo,
QCompDesc FwCompAddr,
DM3TSCCALLBACK pTspEventHandler,
VOID pTspUserInfo,
HANDLE pIocp,
DWORD CompletionKey)

**Inputs:**
- IN LPDM3COMPTSC pTspInfo
- IN QCompDesc FwCompAddr
- IN DM3TSCCALLBACK pTspEventHandler
- OPTIONAL IN PVOID pTspUserInfo
- OPTIONAL IN HANDLE pIocp
- OPTIONAL IN DWORD CompletionKey
- OPTIONAL IN PSTRING pzMyPhoneNumber

**Outputs:** The Component descriptor pointed to by pTspInfo has its members initialized.

**Returns:** SUCCESS or an error code

**Includes:**

**Category:**

**Mode:** SYNC or ASYNC
**Internal Operation**

This function will initialize the TSC component object. A pointer to allocated memory for the object must be provided. The function will perform some TSC initialization, then will call the frameworks Dm3CompINIT() function to initialize the rest of the data structure. Internally an MPATH will be created and associated with the I/O completion port handle passed in. This MPATH will be used exclusively with this instance, and every call to this function will result in a different MPATH device being created.

This function will also discover the bearer channel (timeslot) and line ID associated with this TSC if any, and populate the associated fields in the component data structure. This may be useful to telephony app developers who are used to organizing their data structures along these lines.

<table>
<thead>
<tr>
<th>PDM3COMPTSC</th>
<th>pTspInfo</th>
<th>A pointer to the TSC information structure to be initialized.</th>
</tr>
</thead>
<tbody>
<tr>
<td>QCompDesc</td>
<td>FwCompAddr</td>
<td>Address of the TSC instance object to be initialized</td>
</tr>
<tr>
<td>PVOID</td>
<td>pTspUserInfo</td>
<td>A general purpose pointer for linkage to user app data/objects</td>
</tr>
<tr>
<td>DM3TSCCALLBACK</td>
<td>pTspEventHandler</td>
<td>Event handler function for all events on this instance.</td>
</tr>
<tr>
<td>HANDLE</td>
<td>pIocp</td>
<td>I/O completion port to associate device stream with.</td>
</tr>
<tr>
<td>DWORD</td>
<td>CompletionKey</td>
<td>I/O completion key to associate with this instance.</td>
</tr>
</tbody>
</table>

**Limitations/Assumptions**

1. Not all members of the DM3COMPTSC structure are provided as arguments to the init function.
2. See Reference 8 for additional details on framework data structure initialization.

**Error Codes**

TBD.
Dm3TscCleanUp() Releases the resources and handles owned by a TSC instance.

Purpose

This function will release the resources owned by a TSC object for use back into the system.

Function Signature

<table>
<thead>
<tr>
<th>Name:</th>
<th>BOOL Dm3TscReleaseResources(PDM3COMPTSC pTspInfo)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs:</td>
<td>IN PDM3COMPTSC pTspInfo</td>
</tr>
<tr>
<td>Outputs:</td>
<td>Releases the resources allocated to the component</td>
</tr>
<tr>
<td>Returns:</td>
<td>A valid pointer or NULL if failure</td>
</tr>
<tr>
<td>Includes:</td>
<td></td>
</tr>
<tr>
<td>Category:</td>
<td></td>
</tr>
<tr>
<td>Mode:</td>
<td>SYNC</td>
</tr>
</tbody>
</table>

Internal Operation

This function will release the MPATH that was associated with this instance, by closing it’s handle. In addition if an active call is associated with this instance, a “release call” command will be issued, and the call ID reset to its default value in the data structure.

Limitations/Assumptions

1. While no function currently exists in Windows to detach the MPATH from the IO completion port, closing it effectively accomplishes this. This functionality may be explicitly available in NT 5.0

Error Codes

TBD.
TSC Command Functions

Dm3TscMakeCall() Places an outgoing call

Purpose

This function will result in an outgoing call being initiated on the specified TSC instance and a call ID being created and returned to the application. This function allocates, prepares, and sends the MMB for the TSC standard message TSC_MsgMakeCall.

Function Signature

<table>
<thead>
<tr>
<th>Name:</th>
<th>BOOL Dm3TscMakeCall (LPDM3COMPTSC, pTspInfo, LPSTR szDestAddress, LPSTR szOriAddress, BOOL fCallProg, LPINFOELEMENT pInfoArray)</th>
</tr>
</thead>
</table>
| Inputs: | IN LPDM3COMPTSC pTscInfo  
IN LPSTR szDestAddress  
IN LPSTR szOriAddress  
IN BOOL fCallProg  
OPTIONAL IN LPINFOELEMENT pInfoArray |
| Outputs: | Call is initiated |
| Returns: | SUCCESS or an error code |
| Includes: | |
| Category: | |
| Mode: | SYNC or ASYNC |

Argument Descriptions

- pTscInfo: TSC Host component information structure
- pzDestAddress: Phone number to be dialed
IPTGate Demo
(IP - PSTN Gateway)
User's Guide

pzOrigAddress  My phone Number. Used as “Originating” address.
FCallProg     Flag controlling the enablement of call progress.
PInfoArray    Pointer to an array of information elements that can provide additional data to the make call command payload for protocols such as ISDN and VOIP.

Internal Operation

This function will prepare the message and payload necessary for the TSC_MsgMakeCall message. The payload structure consists of three essential elements, all of which are exposed through this functions as shown below:

<table>
<thead>
<tr>
<th>Destination Address</th>
<th>Set to value of szDestAddress specified in argument list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Originating Address</td>
<td>Set to value of szOrigAddress from argument list</td>
</tr>
<tr>
<td>Call Progress</td>
<td>Set to state requested by value of fCallProg in argument list, either enabled or disabled</td>
</tr>
</tbody>
</table>

In addition to these required values, the array pInfoArray is a list of KV sets which may be added to the MakeCall command. If this pointer is NULL, no KV set processing will be performed and the standard TSC_MsgMakeCall_t payload will be sent down.

As Call ID is not available until generation of the TSC_MsgMakeCallCmplt event, the handler for this event must update this field in the objects data structure.

Limitations/Assumptions

This function has the following limitations:
1. The TSC_MsgMakeCallCmplt message must be handled by the application when ASYNC operation is specified.

2. Assumes the Mpath was opened with FILE_FLAG_OVERLAPPED specifier if ASYNC operation specified.

Error Codes

Standard Error Messages (See Section on Error Handling).
Dm3TscAnswerCall() Seizes the line of an incoming call

Purpose

This function will answer the call of any bearer channel in response to a call offering. The set up and receipt of incoming call events is beyond the scope of this function and is assumed to be provided elsewhere.

Function Signature

**Name:** BOOL Dm3TscAnswerCall (LPDM3COMPTSC, pTspInfo, UINT RingCount, LPINFOELEMENT pInfoArray)

**Inputs:**
- pTspInfo • A pointer to the TSC information structure
- RingCount • The number of rings to wait before answering
- pInfoArray • An option array of KV sets that will supplement the normal Answer call command payload.

**Outputs:** None

**Returns:** SUCCESS or FAIL

**Includes:**

**Category:**

**Mode:** SYNC

Internal Operation

This function will prepare the message and payload necessary for the TSC_MsgAnswerCall message. The payload consists of 2 elements, one of which is exposed through this function as shown below:
Appendix E

<table>
<thead>
<tr>
<th>Call Id</th>
<th>Handle to the call to be answered. This would have been provided as data in the “offered” event and stored as part of the component instance descriptor structure (DM3COMPTSC).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Rings</td>
<td>The number of rings to wait before answering</td>
</tr>
<tr>
<td>pInfoArray</td>
<td>An Optional pointer to an array of KV sets.</td>
</tr>
</tbody>
</table>

In addition to these required values, the array pInfoArray is a list of KV sets which may be added to the AnswerCall command. If this pointer is NULL, no KV set processing will be performed and the standard TSC_MsgAnswerCall_t payload will be sent down.

Only SYNC operation is supported as there is no reply message associated with this command. The only indication that this command succeeded is that a Std_MsgEvtDetected will come back with a TSCEvtCallState showing the transition to the “connected” state.

Limitations

This function assumes that Std_MsgEvtDetected event has been enabled for a call state transition to the “connected” state or it will not be possible for the user to ascertain when the line has actually been seized.

Error Codes

Standard Error Messages (See Section on Error Handling).
Dm3TscAnswerCall( ) Seizes the line of an incoming call

Purpose

This function will answer the call of any bearer channel in response to a call offering. The set up and receipt of incoming call events is beyond the scope of this function and is assumed to be provided elsewhere.

Function Signature

**Name:** BOOL Dm3TscAnswerCall ( LPDM3COMPTSC, pTspInfo, UINT RingCount, LPINFOELEMENT pInfoArray)

**Inputs:**
- **pTspInfo**
  - A pointer to the TSC information structure
- **RingCount**
  - The number of rings to wait before answering
- **pInfoArray**
  - An option array of KV sets that will supplement the normal Answer call command payload

**Outputs:** None

**Returns:** SUCCESS or FAIL

**Includes:**
- Category: ...
- Mode: SYNC

Internal Operation

This function will prepare the message and payload necessary for the TSC_MsgAnswerCall message. The payload consists of 2 elements, one of which is exposed through this function as shown below:
<table>
<thead>
<tr>
<th>Call Id</th>
<th>Handle to the call to be answered. This would have been provided as data in the “offered” event and stored as part of the component instance descriptor structure (DM3COMPTSC).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Rings</td>
<td>The number of rings to wait before answering</td>
</tr>
<tr>
<td>pInfoArray</td>
<td>An Optional pointer to an array of KV sets.</td>
</tr>
</tbody>
</table>

In addition to these required values, the array pInfoArray is a list of KV sets which may be added to the AnswerCall command. If this pointer is NULL, no KV set processing will be performed and the standard TSC_MsgAnswerCall_t payload will be sent down.

Only SYNC operation is supported as there is no reply message associated with this command. The only indication that this command succeeded is that a Std_MsgEvtDetected will come back with a TSCEvtCallState showing the transition to the “connected” state.

**Limitations**

This function assumes that Std_MsgEvtDetected event has been enabled for a call state transition to the “connected” state or it will not be possible for the user to ascertain when the line has actually been seized.

**Error Codes**

Standard Error Messages (See Section on Error Handling).
Dm3TscAcceptCall() Accepts an incoming call

Purpose

This function will accept the call of any bearer channel in response to a call offering. This is different than outright answering a call in that the call state is moved to the “accepted” state as opposed to the “connected” state. This may have implications with regard to CO communication dependent upon the protocol in use.

The set up and receipt of incoming call (“offered”) events is beyond the scope of this function and is assumed to be enabled elsewhere.

Function Signature

<table>
<thead>
<tr>
<th>Name:</th>
<th>PVOID Dm3TscAcceptCall (LPDM3COMPTSC, pTspInfo)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs:</td>
<td>LPDM3COMPTSC pTspInfo</td>
</tr>
<tr>
<td>Returns:</td>
<td>SUCCESS or error code if failure</td>
</tr>
<tr>
<td>Includes:</td>
<td>• A pointer to the TSC information structure</td>
</tr>
<tr>
<td>Category:</td>
<td>•</td>
</tr>
<tr>
<td>Mode:</td>
<td>SYNC</td>
</tr>
</tbody>
</table>

Internal Operation

This function will use the convenience functions to prepare the message and payload necessary for the TSC_MsgAcceptCall message. The payload consists of 1 element as shown below:

<table>
<thead>
<tr>
<th>Call Id</th>
<th>Handle to the call to be accepted. This would have been provided as data in the “offered” event.</th>
</tr>
</thead>
</table>
Only SYNC operation is supported as there is no reply message associated with this command. The only indication that this command succeeded is that a Std_MsgEvtDetected will come back with a TSCEvtCallState showing the transition to the “Accepted” state.

Limitations

This function assumes that Std_MsgEvtDetected event has been enabled for a call state transition to the “accepted” state or it will not be possible for the user to ascertain when the command is actually done.

Error Codes

Standard, see Section on error codes.
Dm3TscDropCall()  Drops a call

Purpose

This function “hangs up” a particular call but leaves the call identifier open to support post call queries.

Function Signature

Name:  PVOID  Dm3TscDropCall(LPDM3COMPTSC, pTspInfo)
Inputs:  LPDM3COMPTSC  • A pointer to the TSC information structure  
        pTspInfo
Returns:  SUCCESS or error code if failure
Includes:  
Category:  
Mode:  SYNC

Internal Operation

This function will use the convenience functions to prepare the message and payload necessary for the TSC_MsgDropCall message. The payload consists of 1 element as shown below:

<table>
<thead>
<tr>
<th>Call Id</th>
<th>Handle to the call to be dropped.</th>
</tr>
</thead>
</table>

Only SYNC operation is supported as there is no reply message associated with this command. The only indication that this command succeeded is that a Std_MsgEvtDetected will come back with a TSCEvtCallState showing the transition to the “idle” state.
Limitations

Error Codes

This function will return an error if the Call ID is invalid. All other errors are returned through the STD_MsgError event as detailed in section 6.
Dm3TscReleaseCall() Releases the call ID

Purpose

This function will initiate the release the call ID specified by issuing the TSC_MsgReleaseCall command.

Function Signature

Name: PVOID Dm3TscReleaseCall(LPDM3COMPTSC, pTspInfo)
Inputs: LPDM3COMPTSC
pTspInfo
• A pointer to the TSC information structure
Returns: SUCCESS or error code if failure
Includes: •
Category:
Mode: TBD

Internal Operation

This function will use the convenience functions to prepare the message and payload necessary for the TSC_MsgReleaseCall message. The payload consists of 1 element as shown below:

<table>
<thead>
<tr>
<th>Call Id</th>
<th>Handle to the call to be Released.</th>
</tr>
</thead>
</table>

Only SYNC operation is supported as there is no reply message associated with this command. The only indication that this command succeeded is that a Std_MsgEvtDetected will come back with a TSC_EvtCallState showing the transition to the “Null” state. If the call has not already been dropped, then this function will first cause a transition to the “idle” state, then the null state transition.
Appendix E

Limitations
None.

Error Codes

See Standard Tsc Error Codes section.
Dm3TscRejectCall( ) Rej...an incoming call

Purpose

This function allows the user to reject an incoming call. A reason for this rejection can be provided to allow the proper setting of protocol dependent information. The function prepares the message and payload for the TSC_MsgRejectCall command.

Function Signature

Name: PVOID Dm3TscRejectCall( LPDM3COMPTSC, pTspInfo, UINT Reason)

Inputs:
- LPDM3COMPTSC
- pTspInfo
- UINT Reason

• A pointer to the Tsc information structure
• The reason for the rejection, BUSY, CONGESTION, or UNAVAILABLE

Returns: SUCCESS or error code if failure

Includes:

Category: TBD

Mode: TBD

Internal Operation

This function prepares the command and payload for the TSC_MsgRejectCall command. This payload contains a reason for the rejection so that certain protocols can transmit this information.

Limitations

See IPLink Reference Guide on TSC_MsgRejectCall
Error Codes

Standard error messages.
Dm3TscGetChanState( ) Retrieves channel state information.

Purpose

This function will retrieve the channel state associated with the particular component specified. This allows the user to manage call states in his application.

Function Signature

Name: PVOID Dm3TscGetChanState(LPDM3COMPTSC, pTspInfo, PUINT ChanState)

Inputs:
- LPDM3COMPTSC
- pTspInfo
- PUINT ChanState

• A pointer to the Tsc information structure
• The state of the channel is returned here

Returns:
- SUCCESS or error code if failure

Includes:

Category: Mode: Synchronous or Asynchronous

Internal Operation

This function is directly associated with the TSC_MsgGetChanState command. It will internally handle the formatting of the command.

In SYNC mode, the function will block until the TSC_MsgGetChanStateCmplt reply is received. It will parse this data structure and return the current channel state in the variable indicated.

Note that use of this function will NOT update the data field associated with channel state in the TSC component descriptor structure.
In Async mode, the …Cmplt event will be generated, and the event handler for this message will take care of returning the channel state.

Limitations

None.

Error Codes

Standard.
Dm3TscGetCallState( ) Gets the current call state

Purpose

This function will set up the message and payload for the TSC_MsgGetCallState command.

Function Signature

Name: PVOID Dm3TscGetCallState (LPDM3COMPTSC, pTspInfo, UINT CallState)

Inputs:
- LPDM3COMPTSC
- pTspInfo
- UINT CallState

Includes:
- A pointer to the Tsc information structure
- The state of the call is returned here

Returns:
- SUCCESS or error code if failure

Category: 

Mode: Synchronous or Asynchronous

Internal Operation

This function is directly associated with the TSC_MsgGetCallState command. It will internally handle the formatting of the command.

In SYNC mode, the function will block until the TSC_MsgGetCallStateCmplt reply is received. It will parse this data structure and return the current call state in the variable indicated.

In Async mode, the ...Cmplt event will be generated, and the event handler for this message will take care of returning the call state.
Appendix E

Limitations

None.

Error Codes

Standard. See section on error codes.
Dm3TscEvtHndlr() Main, top level handler for all Tsc events.

Purpose

To provide a single callback handler for Tsc events on an instance.

Function Signature

Name: PVOID Dm3TscEvtHndlr (PDM3COMP pTspInfo, ULONG ulMsgType, QMsgRef pReplyMsg)

Inputs:
- PDM3COMP pTspInfo • A pointer to the generic component information
- ULONG ulCmdType • The last command associated with this event. This is important to provide a context for StdMsgError processing
- ULONG ulMsgType • The particular event being returned
- QMsgRef pReplyMsg • A pointer to any event data

Returns: SUCCESS or error code if failure

Includes:

Category: N/A

Mode: N/A

Internal Operation

This function is the top level event and message dispatcher for all Tsc component events. It is basically a large switch statement that passes specific event information to individual handers.
Appendix E

Limitations

None.

Error Codes

Standard. See section on error codes.
Event Handling

OnTscMakeCallCmplt( ) Event Handler for TSC_MakeCallCmplt message

Purpose

To provide message processing for the TSC_MsgMakeCallCmplt message.

Function Signature

Name: LPVOID OnTscMakeCallCmplt (PDM3COMP pTspInfo, ULONG ulCmdtType, ULONG ulMsgType, QMsgRef pReplyMsg)

Inputs:
- PDM3COMP pTspInfo • A pointer to the generic component information
- ULONG ulCmdtType • The Command that this Cmplt is for, (should be TSC_MsgMakeCall)
- ULONG ulMsgType • The particular event being returned
- QMsgRef pReplyMsg • A pointer to any event data

Returns:
- A Pointer to the base component descriptor structure of the client for this event

Includes:

Category: N/A

Mode: N/A

Internal Operation

This function handles the above message by processing the payload (CallID) and saving it into the CallId field of the Tsc component data structure.
Appendix E

Limitations

• This function should not initiate any synchronous command calls directly or it will create recursion through the library

Error Codes

Standard. See section on error codes.
OnTscStdMsgEvtDetected() Handler for STD_MsgEvtDetected events on a Tsc instance.

Purpose

To provide message processing for the Std_MsgEventDetected message which for TSC, implies receipt of one of the three types of unsolicited messages.

Function Signature

Name: LPVOID OnTscStdMsgEvtDetected (PDM3COMP pTspInfo, ULONG ulCmdType, ULONG ulMsgType, QMsgRef pReplyMsg)

Inputs:
- PDM3COMP pTspInfo: A pointer to the generic component information
- ULONG ulCmdType: The Command that this Cmplt is for. (should be TSC_MsgMakeCall)
- ULONG ulMsgType: The particular event being returned
- QMsgRef pReplyMsg: A pointer to any event data

Returns:
- A void pointer the application can use to return any desired data

Includes:

Category:

Mode: N/A

Internal Operation

This function processes the payload to determine which of the three types of TSC messages have been returned and then calls the appropriate sub handler.
Appendix E

Limitations

• This function should not initiate any synchronous framework command calls directly or it will create recursion through the library

Error Codes

Standard. See section on error codes.
OnTscCallInfoEvent( ) Handler for Call Information events on a Tsc instance.

Purpose

To provide message processing for call information events received by the instance. For a complete list of these events see the documentation in the TSC user’s guide.

Function Signature

Name: LPVOID OnTscCallInfoEvent (PDM3COMP pTspInfo, ULONG ulCmdType, ULONG ulMsgType, QMsgRef pReplyMsg)

Inputs:
- PDM3COMP pTspInfo • A pointer to the generic component information
- ULONG ulCmdType • The Command associated with this event
- ULONG ulMsgType • The particular event being returned

Returns:
- QMsgRef pReplyMsg • A pointer to any event data
- A Pointer to the base component descriptor structure of the client for this event

Includes:

Category: N/A

Mode: N/A

Internal Operation

This function processes the following possible call information events as outlined in the TSC Resource User's Guide.
Appendix E

- CallAnalysis Gives Post call analysis results (PAMD, FAX, PVD, etc..)
- CallerCategory
- CallerId
- CallerIdType
- Display
- Language

The payload is processed to obtain these, and any data associated with these elements. Code will be implemented to show the removal of this data, but actual saving of this info to persistent storage is left to the user.

Limitations

- This function should not initiate any synchronous framework command calls directly or it will create recursion through the library
- No constructs are provided for the storage of Call Progress and Caller ID information outside the lifetime of this function.

Error Codes

Call ID returned as payload must match Call ID currently saved with component or error will be flagged.
**OnTscChanStateEvent( )** Handler for *Channel State transition* events on a Tsc instance.

**Purpose**

To provide message processing for channel state events received by the instance.

**Function Signature**

```plaintext
Name: LPVOID OnTscChanStateEvent (PDM3COMP pTspInfo, ULONG ulMsgType, QMsgRef pReplyMsg)

Inputs:
PDM3COMP pTspInfo • A pointer to the generic component information
ULONG ulMsgType • The particular event being returned
QMsgRef pReplyMsg • A pointer to any event data

Returns:
A Pointer to the base component descriptor structure of the client for this event

Includes:

Category: N/A
Mode: N/A
```

**Internal Operation**

This function processes all the possible channel state transition events as outlined in Appendix K of the Tsc user’s guide.

The payload is processed to obtain these, and any data associated with these elements. Code will be implemented to show the removal of this data, but actual saving of this info to persistent storage is left to the user.
Appendix E

Limitations

• This function should not initiate any synchronous framework command calls directly or it will create recursion through the library

• No constructs are provided for the persistent storage of channel state information, this is left to the user.

Error Codes

Call ID returned as payload must match Call ID currently saved with component or error will be flagged.
OnTscCallStateEvent() Handler for Call State transition events on a Tsc instance.

**Purpose**

To provide message processing for call state events received by the instance.

**Function Signature**

```plaintext
Name: LPVOID OnTscCallStateEvent (PDM3COMP pTspInfo, ULONG ulMsgType, QMsgRef pReplyMsg)
```

**Inputs:**
- PDM3COMP pTspInfo: A pointer to the generic component information
- ULONG ulMsgType: The particular event being returned
- QMsgRef pReplyMsg: A pointer to any event data

**Returns:**
A Pointer to the base component descriptor structure of the client for this event

**Internal Operation**

This function processes all the possible call state transition events as outlined in Appendix J of the TSC User’s Guide.

The payload is processed to the reason for the call state transition, and any data associated with these elements. Code will be implemented to show the removal of this data, but actual saving of this info to persistent storage is left to the user.

Payload processing is as follows:
Label: Ignored. Related to RTC stuff which I don’t care about since this is host notification

Type: Ignored. Related to above.

CallId: Checked against active callID of current instance.

CallState: The new call state is stored in the Tsc data structure.

Reason: Processed locally.

Limitations

- This function should not initiate any synchronous framework command calls directly or it will create recursion through the library
- No constructs are provided for the persistent storage of channel state reason information, this is left to the user.

Error Codes

Call ID returned as payload must match Call ID currently saved with component or error will be flagged.
**NetTSC Functions**

**Dm3NetTscInit**

- **Name:** Dm3NetTscInit
- **Inputs:** None
- **Outputs:** DM3 Net TSC
  - Component initialized
- **Returns:** TRUE if successful
- **Includes:**
- **Category:**
- **Mode:**

Initializes the DM3 Net TSC Component
NetTSCResetSession

**Name:** NetTSCResetSession

**Inputs:**
- `USHORT ch` • Channel to reset
- `BOOL clear` • Boolean to know if reset totally
  - `TRUE` • clear totally
  - `FALSE` • clear the related fields of this call

**Outputs:** None

**Returns:** Void

**Includes:**

**Category:**

**Mode:**

Resets session structure to default values
### Dm3NTscNonStdCmd

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Dm3NTscNonStdCmd</td>
</tr>
<tr>
<td>Inputs</td>
<td>lpTsc</td>
</tr>
<tr>
<td></td>
<td>• A pointer to the TSC instance data structure</td>
</tr>
<tr>
<td></td>
<td>unReason</td>
</tr>
<tr>
<td></td>
<td>• The reason the call is being dropped</td>
</tr>
<tr>
<td>Outputs</td>
<td>None</td>
</tr>
<tr>
<td>Returns</td>
<td>DM3SUCCESS or DM3FAIL</td>
</tr>
<tr>
<td></td>
<td>• If DM3FAIL, use GetLastError()</td>
</tr>
<tr>
<td>Includes</td>
<td>•</td>
</tr>
<tr>
<td>Category</td>
<td>•</td>
</tr>
<tr>
<td>Mode</td>
<td>•</td>
</tr>
</tbody>
</table>

Command initiation function for TSC_MsgDropCall
Appendix E

Dm3NTscUll

**Name:** Dm3NTscUll

**Inputs:**
- lpTsc
- unReason
  - A pointer to the TSC instance data structure
  - The reason the call is being dropped

**Outputs:** None

**Returns:** None
  - DM3SUCCESS or DM3FAIL. If DM3FAIL, use GetLastError().

**Includes:**

**Category:**

**Mode:**

Command initiation function for TSC_MsgDropCall
### NETTSCClusterInit

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: NETTSCClusterInit</td>
<td></td>
</tr>
<tr>
<td>Inputs:</td>
<td></td>
</tr>
<tr>
<td>IpNETTSCCluster</td>
<td>pointer to an allocated NETTSCCLUSTER object</td>
</tr>
<tr>
<td>ucBoardNum</td>
<td>board number, this cluster should be allocated</td>
</tr>
<tr>
<td>ucLineId</td>
<td>Network line# identifying the cluster</td>
</tr>
<tr>
<td>ucChanId</td>
<td>Channel within the given line# identifying the cluster</td>
</tr>
<tr>
<td>lpNETTSCClusterCallback</td>
<td>handler called when any of the qvscluster events are detected.</td>
</tr>
<tr>
<td>lpCriticalSection</td>
<td>critical section in case of multiThread</td>
</tr>
<tr>
<td>hIOPC</td>
<td>handle to the IO Completion port for async parms</td>
</tr>
<tr>
<td>dwIOPKey</td>
<td>the related I/O completion key</td>
</tr>
</tbody>
</table>

| Outputs:     | None |
| Returns:     | Success or fail |
| Includes:    |                |
| Category:    |                |
| Mode:        |                |

Initialize and allocate a NETTSCCLUSTER object. If successful, the method completes with NETTSCCLUSTEREVENT_INITCMPLT.
NETTSCClusterGetAllComps

| Name:       | NETTSCClusterGetAllComps |
| Inputs:     | lpNETTSCCluster • a pointer to an initialized NETTSCCluster object |
| Outputs:    | None • |
| Returns:    | Success or fail • |
| Includes:   | • |
| Category:   | |
| Mode:       | |

Get all the components allocated into a NETTSCCluster. If successful, the method completes with NETTSCCLUSTEREVENT_LISTENCMLT.

Initiate Allocation of the cluster identified by the lpNETTSCCluster object and if allocation is successful Get all the components in the cluster.
**NETTSCClusterListen**

- **Name:** NETTSCClusterListen
- **Inputs:**
  - lpNETTSCCluster: pointer to an allocated DM3CLUSTER object
  - unTimeSlot: a CTBUS timeslot number to listen to
- **Outputs:** NONE
- **Returns:** Success or fail
- **Includes:**
- **Category:**
- **Mode:**
- **Cautions:** lpCluster should be an allocated cluster

Initiate the CTBus port in the given cluster to listen to the given timeslot.

If successful, the method completes with NETTSCCLUSTEREVENT_LISTENCMLT.
NETTSCClusterUnlisten

**Name:** NETTSCClusterUnlisten  
**Inputs:**  
- lpNETTSCCluster  
  • pointer to an allocated DM3CLUSTER object

**Outputs:** NONE  

**Returns:** Success or fail  

**Includes:**  

**Category:**  

**Mode:**  

**Cautions:** lpCluster should be an allocated cluster

Initiate the CTBus port in the given cluster to listen to the given timeslot.

If successful, the method completes with NETTSCCLUSTEREVENT_UNLISTENCMLT.
NETTSCClusterGetXmitSlot

**Name:** NETTSCClusterGetXmitSlot()

**Inputs:**
- lpNETTSCCluster: pointer to an allocated DM3CLUSTER object

**Outputs:** None

**Returns:** Success or fail

**Includes:**
- ...

**Category:**

**Mode:**

**Cautions:** lpNETTSCCluster should be an initialized cluster

Initiate the CTBus port in the given cluster to get the assigned transmit timeslot.

If successful, the method completes with NETTSCCLUSTEREVENT_GETXMITSLOTCMPLT.
NETTSCClusterRelease

**Name:** NETTSCClusterRelease()

**Inputs:**
- lpNETTSCCluster • pointer to an allocated NETTSCCLUSTER object

**Outputs:** None

**Returns:** Success or fail

**Includes:**
- •

**Category:**
- •

**Mode:**
- •

**Cautions:** None

Initiate freeing of the allocated NETTSCCluster

---

125
NETTSCClusterCleanup

**Name:** NETTSCClusterCleanup()

**Inputs:**
- lpNETTSCCluster • pointer to an initialized NETTSCCLUSTER object

**Outputs:** None

**Returns:** Success or fail

**Includes:**

**Category:**

**Mode:**

**Cautions:** None

Cleanup all the structures bound to this Cluster. Free all the memory allocated for this structure.
NETTSCClusterGetComponent

<table>
<thead>
<tr>
<th>Name</th>
<th>NETTSCClusterGetComponent()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>lpNETTSCCluster</td>
</tr>
<tr>
<td></td>
<td>• pointer to an allocated</td>
</tr>
<tr>
<td></td>
<td>• NETTSCCLUSTER object</td>
</tr>
<tr>
<td>Outputs</td>
<td>None</td>
</tr>
<tr>
<td>Returns</td>
<td>Success or fail</td>
</tr>
<tr>
<td>Includes</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td></td>
</tr>
<tr>
<td>Mode</td>
<td></td>
</tr>
<tr>
<td>Cautions</td>
<td>None</td>
</tr>
</tbody>
</table>

Initiate getting a component from the cluster depending on the number of components found.
NETTSCClusterGotComponent

**Name:** NETTSCClusterGotComponent()

**Inputs:**
- lpNETTSCCluster: pointer to the NETTSCCluster object
- qcdCompAddr: fw address of the component found in the given cluster. The ucNumCompsFound defined in the lpNETTSCCluster indicates the component (TSC, Tonegen,...) that this fw address identifies

**Outputs:** None

**Returns:** Success or fail

**Includes:**
- None

**Category:**
- None

**Mode:**
- None

**Cautions:** None

Method called when a component is found in the Cluster
# Index

## A
- ANI features, 49
- Answering a call, 21, 42
- Application Foundation Code, 67

## B
- Billing, 49

## C
- Call connection, 15
- Call duration time, 45
- Call establishment, 12
- Call information, 38, 45
- Call progress tones, 12, 49
- Call Status Transition event, 27
- Coder, 16, 42
- Configuration file, 11, 23, 25, 42
- Convenience functions, 23

## D
- D/160LS, 15
- Data structures, 46
- DE_LCOFF, 35, 36
- DE_LEOFF, 44
- DE_RINGS, 33
- DE_TONEON, 35, 36, 44
- Debugging functions, 23
- Destination address, 49
- Directory service, 49
- Disconnect detection, 28
- Disconnect indication, 12
- Disconnect supervision, 22
- DM_LCOFF, 27
- DM_RINGS, 27
- DM3_IPT_KEY, 28
- Dm3sp.c, 23
- DTMF, 49
- Dual frequency cadence tone, 28
- Duration time, 38
- dx_addtone(), 28
- dx_blddtcad(), 28
- dx_deltones(), 27
- dx_open(), 27
- dx_setevtmск(), 27
- dx_sethook(), 27
- dx_setrings(), 27

## E
- Error functions, 23
- Exiting a Call, 39

## F
- Functions
  - dx_addtone(), 28
  - dx_blddtcad(), 28
dx_deltones(), 27
dx_open(), 27
dx_setevtmsk(), 27
dx_sethook(), 27
dx_setrings(), 27
GetQueuedCompletionStatus(), 28, 29
IPTOpenNetTSC(), 26
pstnOpenFrontEnd(), 27
sr_getevtdatap(), 28
sr_getevtdev(), 28
sr_getevttype(), 28

G
Gatedbg.c, 23
Gatedbg.h, 24
Gatedefs.h, 24
Gateipt.c, 26
Gatemain.c, 23, 29
Gatepars.c, 23
Gatepstn.c, 23
Gatestat.c, 24
Gatestrc.h, 24
Gatevars.h, 24
GetQueuedCompletionStatus(), 28, 29

H
H.323 drop indication, 12
H.323 terminal, 11, 22

I
I/O completion port, 25, 28
I/O Completion Ports, 67
IP address, 11, 16, 49

IP Inbound Call
State diagram, 31
IPLink initialization procedure, 26
IPTGate.cfg, 11, 12, 16, 23, 25
Iptgate.mak, 25
Iptgate.mdp, 25
IVR system, 49

L
Local phone number, 12, 16
Loop current drop, 12, 22

M
Making a PSTN Outbound Call, 43
MMB structure, 28

N
NetTSC component, 33, 35, 36, 41, 42, 44, 45
NetTSC_H245Data_Type_NonStdCmd,
36, 44
NetTSC_H245Data_Type_UserInputIn
dication, 37, 44
NetTSP cluster, 15

P
PABX, 16
Parsfp.h, 25
PSTN, 9
PSTN Inbound Call
State diagram, 29
PSTN initialization procedure, 27
PSTN Off-hook, 43
Pstnfp.h, 24
pstnOpenFrontEnd(), 27
R
Remote phone number, 11, 16
Retrieving IPT events, 28
Retrieving SRL events, 28
Routing, 11, 43
RTCP info, 38, 45
S
SCbus, 9, 12, 43
SCbusClockMaster, 15
Session log, 20
Source code, 23
sr_getevtdatap(), 28
sr_getevtdev(), 28
sr_getevtttype(), 28
SRAM out time, 16
SRL events, 25
SRL_KEY, 28
State diagrams, 29
IP Inbound Call, 31
PSTN Inbound Call, 29
State machine, 24, 25, 29
Statfp.h, 25
T
TSC component, 38, 45
TSC_EvtCallState_Type_Connected, 35, 43
TSC_EvtCallState_Type_Disconnect, 36, 44
TSC_EvtCallState_Type_Failed, 35, 43
TSC_EvtCallState_Type_Idle, 38, 45
TSC_EvtCallState_Type_Null, 39, 46
TSC_EvtCallState_Type_Offered, 41
TSC_MsgAnswerCall, 41, 42
TSC_MsgDropCall, 36, 37, 44, 45
TSC_MsgGetCallInfo, 41, 42, 43, 45
TSC_MsgMakeCall, 33
TSC_MsgReleaseCall, 38, 45
U
User-defined tones, 28
W
WAIT_FOR_CALL state, 29, 33, 39, 41, 46
WAIT_FOR_CONNECT state, 33, 41
WAIT_FOR_DISCONNECT state, 35, 43, 44
WAIT_FOR_IDLE state, 35, 36, 43, 44
WAIT_FOR_RELEASE state, 38, 45
WinNT, 25