Dialogic®

Application Note
Call Progress Analysis

Call Progress Analysis: Global Call API Usage and Protocol Configuration

Dialogic® System Release 6.0 and Dialogic® Host Media Processing Software Release 2.0 for Windows

Executive Summary

Call Progress Analysis (CPA) is the process of detecting pre-connect information about failed outbound call attempts and the destination party's media type for connected outbound calls. Dialogic® System Release 6.0 and Dialogic® Host Media Processing (HMP) Software Release 2.0 for Windows (or above) provide the flexibility needed to support a broad range of applications that require CPA across all supported technologies and protocols. This application note describes Global Call and associated API usage and protocol configurations recommended for obtaining CPA results with Dialogic® products. A sample test application (sr6callp) is included for exercising CPA scenarios on HMP 2.0, Dialogic interface boards for HMP, and Dialogic boards with DM3 and Springware architectures.

Table of Contents

Introduction

Call progress is an outbound call's pre-connect information, which might include:

- Busy
- No answer
- Circuit Unavailable Special Information Tone (SIT)

Call analysis is the post-connect information about the destination party's media type. This might include:

- Voice
- Answering machine
- Fax machine

The process of performing both call progress and call analysis is generally referred to as call progress analysis (CPA). CPA is typically used in outbound predictive dialling applications for accurate positive voice detection (PVD) or positive answering machine detection (PAMD) after a call connects.

The Global Call API provides a common interface for network-enabled applications regardless of the signaling protocol used, but Global Call's CPA capabilities depend on the signaling protocol and the underlying board technology. In some cases Voice API functions must be used to obtain post-connect information deduced from the audio carried by the bearer channel.

CPA improvements were introduced with Dialogic® System Release 6.0 and Dialogic® Host Media Processing Software Release 2.0 for Windows, which include:

- Multiple updated Perfect Call CPA templates for PAMD and PVD
- Programmatic access to CPA templates at runtime
- Programmatic access to CPA settings on a per call and per channel basis

This application note presents API usage for performing CPA with both the Global Call and Voice APIs along with the recommended method and configuration for a variety of protocol and board type combinations. A test application for exercising CPA scenarios with user-supplied response recordings using readily available Dialogic products is described.

The techniques discussed in this application note are applicable to all versions of Dialogic System Release 6.0 versions for Windows and Linux as well as HMP Release 2.0 or higher. The test application (sr6callp) includes project and make files for both.

CPA API Selection for CPA

Two methods for performing CPA with Global Call are available.

- **Global Call CPA** — Global Call API function calls perform CPA using an attached voice device and report CPA results via the GCEV_MEDIADETECTED and GCEV_DISCONNECTED events.
- **Voice API CPA** — The Voice API dx_dial() function performs CPA using a voice resource directly and reports results via the TDX_CALLP event.

API support for CPA varies by protocol and board type as shown in Table 1.

Protocol	Board Type	CPA API Supported
T1/E1 CAS with PDK	DM3 and Springware	Global Call, Voice
T1/E1 ISDN	DM3	Global Call, Voice
T1/E1 ISDN	Springware	Voice
Analog	DM3	Global Call, Voice
Analog with PDK	Springware	Global Call, Voice
DNI/DSI for HMP	DM3	Global Call, Voice
HMP/IP (H.323 and SIP)	DM3	Voice

Table 1. CPA API Support by Protocol and Board Type

Both the Global Call and Voice API CPA methods use the gc_MakeCall() function to initiate the outbound call. The methods differ in how CPA is configured and initiated and how results are reported.

CPA Recommendations

Depending on application requirements, developers can choose to use either CPA method.

- **Global Call** — Easier to implement and preferred where it is supported. The GCEV_MEDIADETECTED event must be enabled and used as the trigger to obtain CPA results.
- **Voice API** — Used when the Global Call CPA method is not supported or if the application requires more flexibility. When CPA options are set on a per-call basis, the Voice API method allows an application the greatest flexibility for controlling CPA.

Global Call API

The Global Call CPA method initiates CPA automatically as part of call setup using gc_MakeCall(). Pre-connect call progress results for failed call attempts are obtained using the Global Call API function gc_ResultInfo() in response to a GCEV_DISCONNECTED event. Post-connect call analysis results are obtained using the Global Call API function gc_GetCallInfo() in response to a GCEV_CONNECTED event, or optionally in response to a GCEV_MEDIADETECTED event.

The voice device used by Global Call must be attached to the network device. This can be accomplished by:

- Opening the voice device and the network device with gc_OpenEx()
- Opening the voice device separately using dx_open() and attaching it to the network device with gc_AttachResouce()/gc_Listen()/dx_listen().

The method used depends on application requirements. If voice devices are pooled and shared across the application, they should be opened separately and attached. Otherwise, it is more convenient to open the voice device along with the network device using gc_OpenEx(). T1/E1 CAS and Analog network devices require that a voice device be attached when making an outbound call and be able to respond to an inbound call. In this case, it is recommended that the voice resource be attached to the network resource at all times.

Configuration of CPA parameters and attributes is dependent on the protocol and board type. A combination of configuration file settings and settings made using API functions may be required. Certain CPA attributes can be set on a per-channel basis using gc_SetParm() or gc_SetConfigData() and on a per-call basis using the GC_MAKECALL_BLK structure parameter in gc_MakeCall(). Additionally, CPA parameters can be set for Springware using gc_LoadDxParm(). Parameters required for controlling the CPA process are described later in this Application Note. For information on additional parameters, refer to the call progress and call analysis sections of the Global Call API Programming Guide and the individual protocol-specific *Global Call Technology User Guides* for the system release software in use.

Opening Global Call Devices

The Global Call API offers a flexible means of opening network and media devices together as a Global Call device. The resource devices and the protocol to be used are specified with a multi-field device name string parameter to gc_OpenEx().

In general, the format of the device name is:

:<key>_<field name>

Available keys values are:

- **P** — protocol_name specifies the protocol to be used
- **N** — network_device_name specifies the board and time slot names
- **V** — voice_device_name specifies the voice board and channel
- **M** — media_device_name specifies the media board and channel

Sample device names are:

T1 CAS	`:P_pdk_us_mf_io:N_dtiB1T1:V_dxxxB1C1`
DM3 Analog	`:P_pdk_us_mf_io:N_dtiB1T1:V_dxxxB1C1`
Springware Analog	`:P_pdk_us_mf_io:V_dxxxB1C1`
DM/IP / HMP	`:P_SIP:N_iptB1T1:M_ipmB1C1 :V_dxxxB1C1`

For additional information, see gc_OpenEx() in the "Function Information" section of the *Global Call API Library Reference*.

Global Call CPA Method

The Global Call CPA method uses Global Call API functions and events exclusively for CPA settings, initiating CPA with the outbound call and obtaining CPA results. CPA settings may be made either on a per-call or per-channel basis. The recommended method is on a per-call basis as this provides the greatest flexibility.

CPA Settings on a Per-channel Basis

CPA settings are made on a per-channel basis using gc_SetParm(). On DM3, gc_SetParm() enables pre-connect call progress. On Springware, pre-connect call progress is always enabled. Here is an example:

```
GC_PARM gcParm;
gcParm.intvalue = GCPV_ENABLE;
if ( gc_SetParm( gcDevh,
                 CPR_CALLPROGRESS,
                 gcParm) != GC_SUCCESS)
{
     // handle error
}
```

On DM3 and Springware, gc_SetParm() is used to enable post-connect call analysis and the GCEV_MEDIADETECTED event. Here is an example:

```
GC_PARM gcParm;
gcParm.intvalue = GCPV_ENABLE;
if ( gc_SetParm( gcDevh,
                 GCPR_MEDIADETECT,
                 gcParm) != GC_SUCCESS)
{
     // handle error
}
```

CPA Settings on a Per-call Basis

CPA settings can be made on a per-call basis by including information in the GC_MAKECALL_BLK parameter of the gc_MakeCall() function. This is the recommended method. The setting differ for boards with DM3 and Springware architectures.

DM3

On boards using DM3 architecture, a GC_PARM_BLK is used to set parameters via the GC_MAKECALL_BLK gclib element. Here is an example:

```
GC _ MAKECALL _ BLK gcMakeCallBlk;
memset( &gcMakeCallBlk, 0, sizeof( gcMakeCallBlk));

GCLIB _ MAKECALL _ BLK gcLibMakeCallBlk;
memset( &gcLibMakeCallBlk, 0, sizeof( gcLibMakeCallBlk));

gcMakeCallBlk.gclib = &gcLibMakeCallBlk;

GC _ PARM _ BLK* gcParmBlk = 0;

int cpaType = GC _ CA _ ENABLE _ ALL;
gc _ util _ insert _ parm _ ref(   &gcParmBlk,
                                   CCSET _ CALLANALYSIS,
                                   CCPARM _ CA _ MODE,
                                   sizeof(int),
                                   &cpaType);

int cpaSpeedValue = PAMD _ ACCU;
gc _ util _ insert _ parm _ ref(   &gcParmBlk,
                                   CCSET _ CALLANALYSIS,
                                   CCPARM _ CA _ PAMDSPDVAL,
                                   sizeof(int),
                                   &cpaSpeedValue);

gcLibMakeCallBlk.ext _ datap = gcParmBlk;

int result = gc _ MakeCall(     gcDevh,
                                   &crn,
                                   destinationAddress,
                                   &gcMakeCallBlk,
                                   timeout,
                                   EV _ ASYNC);

gc _ util _ delete _ parm _ blk( gcParmBlk);

if ( result != GC _ SUCCESS)
{
      // handle error
}
```

Springware

When a board with Springware architecture and the PDK protocol are used, a PDK_MAKECALL_BLK is needed to set parameters via the GC_MAKECALL_BLK cclib element.

Additional DX_CAP parameters can also be set for Springware from a file using gc_LoadDxParm(). The voice channel parameter file used by gc_LoadDxParm() is a text file containing DX_CAP settings. Only settings that override defaults need be included. Here is an example:

```
GC _ MAKECALL _ BLK gcMakeCallBlk;
memset( &gcMakeCallBlk, 0, sizeof( gcMakeCallBlk));

PDK _ MAKECALL _ BLK pdkMakeCallBlk;
memset( &pdkMakeCallBlk, 0, sizeof( pdkMakeCallBlk));

pdkMakeCallBlk.flags = MEDIA _ TYPE _ DETECT;

gcMakeCallBlk.cclib = &pdkMakeCallBlk;

char errMsgbuf[1024];
if ( gc _ LoadDxParm(     gcDevh,
                          “dxchan.vcp”,
                          errMsgbuf,
                          1024) != GC _ SUCCESS)
{
      // handle error
}

int res = gc _ MakeCall( gcDevh,
                          &crn,
                          destinationAddress,
                          &gcMakeCallBlk,
                          timeout,
                          EV _ ASYNC);

if ( result != GC _ SUCCESS)
{
      // handle error
}
```

See gc_LoadDxParm() in the Function Information section of the *Global Call API Library Reference* for details.

CPA Results

On successful outbound call attempts, GCEV_CONNECTED and GCEV_MEDIADETECTED events will be received.

- **GCEV_CONNECTED** — Signals that a connection has been established.
- **GCEV_MEDIADETECTED** — Signals that CPA has completed and result information is available.

If the GCEV_CONNECTED event arrives before the GCEV_MEDIADETECTED event, the result obtained using gc_GetCallInfo() in response to the GCEV_CONNECTED event will be GCCT_INPROGRESS. The CPA result is obtained by using gc_GetCallInfo() upon receiving the GCEV_MEDIA_DETECTED event. Here is an example:

```
METAEVENT metaEvent;
if (gc _ GetMetaEvent(&metaEvent) != GC _ SUCCESS)
{
     // handle error
}

switch (metaEvent.evttype)
{
     …
     case GCEV _ CONNECTED:
     {
            char connectType;
            if ( gc _ GetCallInfo(      crn,
                                        CONNECT _ TYPE,
                                        &connectType) !- GC _ SUCCESS)
            {
                  // handle error;
            }
     }
     break;

     …

     case GCEV _ MEDIADETECTED:
     {
            char mediaDetectedType;
            if ( gc _ GetCallInfo(      crn,
                                        CONNECT _ TYPE,
                                        &mediaDetectedType) !- GC _ SUCCESS)
            {
                  // handle error;
            }
     }
     break;
     …
}
```

Voice API

While using the Voice API method, the application initiates CPA with the dx_dial() function at the earliest stage in call setup where far end audio is available on the bearer channel. Both pre-connect and post-connect results are obtained using ATDX_CPTERM() and ATDX_CONNTYPE() in response to a TDX_CALLP event. The TDX_CALLP event is enabled by issuing a dx_dial() function call and setting the DX_CALLP bit of the mode parameter .

The voice device used for CPA must have the audio path of the network device routed to it before starting CPA using dx_dial(). If the voice device was not opened with the network device using gc_OpenEx(), the routing is performed using the CT Bus routing API appropriate for the device types. When the voice device has been opened along with the network device using gc_OpenEx(), the device handle for the voice device is obtained using gc_GetResourceH().

Configuration of CPA parameters and attributes is dependent on the protocol and the board type. A combination of configuration file settings and settings made using API functions may be required. CPA parameters can be set using the DX_CAP structure and included as a parameter when calling the dx_dial() function. For boards with Springware architecture, the dx_initcallp() function is used to initialize the voice resource for CPA. For more information, see the "Call Progress Analysis" section of the *Voice API Programming Guide* for the System Release in use.

Voice API CPA Method

The Voice API CPA method uses a combination of Global Call and Voice API functions and events. For example, the Global Call API function gc_MakeCall() is used to initiate the outbound call. Voice API functions are used for CPA settings and initiating CPA. CPA is initiated by calling dx_dial() at the first point in call setup where far end audio is available, usually upon receiving the GCEV_ALERTING event. In certain protocols, other events are provided that can start the CPA, such as GCEV_PROCEEDING and GCEV_PROGRESSING. If the GCEV_ALERTING event is not supported by the particular protocol in use, dx_dial() is issued immediately after dialing has completed. Some protocols need a voice resource during the dial process and the attached voice resource cannot be used until the dial process has completed. If the GCDEV_ALERTING and GCDEV_PROCEEDING are not available for these protocols, the application needs to initiate CPA when the GCEV_CONNECTED event is received. Once the call has been established and CPA has been initiated, the results are obtained in response to Voice API events.

CPA Settings

CPA settings are established using the DX_CAP structure which is supplied as a parameter to the dx_dial() function. Not all fields of the DX_CAP structure are supported on boards with both DM3 and Springware architectures. For details for DX_CAP, see the "Data Structures" section of the *Voice API Library Reference*.

For boards with Springware architecure, dx_deltones() and dx_initcallp() must be called to initialize CPA before calling dx_dial() to perform CPA. The dx_deltones() function clears all active tone templates for the channel. The dx_initcallp() function initializes the channel for CPA, which will remain active until dx_deltones() is called again. Note that dx_deltones() clears all global tone definitions (GTD) for the channel.

```
if ( dx_deltones( voxDevh) == -1)
{
     // handle error
}

if ( dx_initcallp( voxDevh) == -1)
{
     // handle error
}
```

If changes are required to the standard CPA tone definitions, they are made prior to calling dx_deltones()/dx_initcallp() and if user-defined tones are used, they are defined with dx_addtone() after calling dx_deltones()/dx_initcallp().

Initiating CPA

CPA is initiated by calling the dx_dial() without a dial string and including DX_CALLP in the mode. CPA settings are made via the DX_CAP structure:

```
DX_CAP cap;      // Voice call analysis & call progress structure
dx_clrcap(&cap);
cap.ca_intflg = DX_PAMDOPTEN;
cap.ca_pamd_spdval = 3;

if (dx_dial( voxDevh, "", &cap, DX_CALLP|EV_ASYNC) != 0)
{
     // handle error
}
```

CPA Results

When the dx_dial() function is used for CPA, the Voice API TDX_CALLP event signals that CPA has completed and results are available. Use the Voice API functions ATDX_CPTERM() and ATDX_CONNTYPE() to determine the CPA termination and connection type as in the following example:

```
METAEVENT metaEvent;
if (gc_GetMetaEvent(&metaEvent) != GC_SUCCESS)
{
     // handle error
}

switch (metaEvent.evttype)
{
     …
     case TDX_CALLP:
     {
            long cpTerm = ATDX_CPTERM( metaEvent.evtdev ) );
            if ( cpTerm == CR_CNCT)
            {
                  long connType = ATDX_CONNTYPE(metaEvent.evtdev )
            }
     }
     break;
     …
}
```

CPA Configuration

For CPA configuration, PDK protocols must be set for boards with either Springware or DM3 architecture. Call progress must be enabled in the appropriate DM3 ISDN config file for boards with DM3 architecture.

PDK Protocols

Three PDK configuration settings are required. These configurations are contained in PDK cdp files in the system release software installation cfg directory. Each protocol has a configuration file. The file for T1, for example, is pdk_us_mf_io.cdp

The ConnectType parameter in a cdp file determines how connects are handled for out-of-band signaling and CPA. The desired behavior is to receive a connect event when a connection is detected from the out-of-band signaling and receive a media-detected event when the media type is detected from CPA. For CAS protocols, the CDP_OUT_ConnectType parameter is set to 1, and for R2 protocols, the CDP_ConnectType parameter is set to 1. Here is an example:

```
All INTEGER _ t CDP _ OUT _ ConnectType = 1
```

The parameters PSL_MakeCall_CallProgress (for boards with Springware architecture) and PSL_CACallProgressOverride (for boards with DM3 architecture) determine call progress operations. The desired behavior is to allow the application to dynamically configure call progress operation. To achieve this, both parameters are set to 2. Here are examples:

```
R4 INTEGER _ t PSL _ MakeCall _ CallProgress = 2
DM3 INTEGER _ t PSL _ CACallProgressOverride = 2
```

The parameters PSL_MakeCall_MediaDetect (for boards with Springware architecture) and PSL_CAMediaDetectOverride (for boards with DM3 architecture) determine call analysis operations. The desired behavior is to allow the application to dynamically configure call analysis operation. To achieve this, both parameters are set to 2. Here are examples:

```
R4 INTEGER _ t PSL _ MakeCall _ MediaDetect = 2
DM3 INTEGER _ t PSL _ CAMediaDetectOverride = 2
```

The default configurations for the above parameters vary by protocol and should be verified for the cdp file being used.

DM3 ISDN

DM3 ISDN protocol configurations are contained in config files located in the system release installation data directory. For example, the file for 5ess on a DM/V960A-4T1 board is ml2_qsa_5ess.config.

Call progress is enabled or disabled with the “CallProgress” parameter, which must be set to “y.”

For example:

```
Variant CallProgress y ! y=Allow call progress, n=disallow
```

When changes are made to a config file, a new fcd file must be generated using the fcdgen utility.

Sample Test Application

The test application available with this application note (sr6callp) is provided as a working example that exercises various CPA scenarios. The sample application can be used as both the stimulus and response for testing various CPA scenarios. The interface has been designed to support a single channel from command-line arguments, or multiple inbound and outbound channels from a configuration file. Both pre-connect and post-connect responses are supported with user-provided response recordings.

Configurable options include:

- Global Call or Voice API (dx_dial) CPA method
- Inbound response played from voice file
- Inbound response played from a list of voice files
- Inbound response when call is offered or connected
- Maximum number of calls to attempt or accept
- Delay between outbound calls

A Zip file containing sr6callp and other components can be downloaded at http://www.dialogic.com/goto?HMP2.

Design Features

The console-based sr6callp application uses an asynchronous programming model and state machines to control application logic. Device state and call state are separated into two state machines using the State Design Pattern. This approach has the advantage that event handling and device control are decoupled from the application call-handling logic. All call-handling commands and events are channelled through the device-state machine to the call-state machine, providing convenient handling of blocked and unblocked events.

Development Environment Support

Project files are supplied for VC++ 6.0 and VC++ 7.1 on Microsoft® Windows, and a make file is supplied for Linux. Both environments require that system release software be installed.

Sample Test Configuration

A convenient way to test CPA functionality in a laboratory environment with digital interface boards is to connect pairs of ports back-to-back with T1 crossover cables. Analog boards must be connected to PBX analog station ports. For SIP with a DM/IP board or an HMP system, only a network connection is required. See Figure 1 for a sample test configuration.

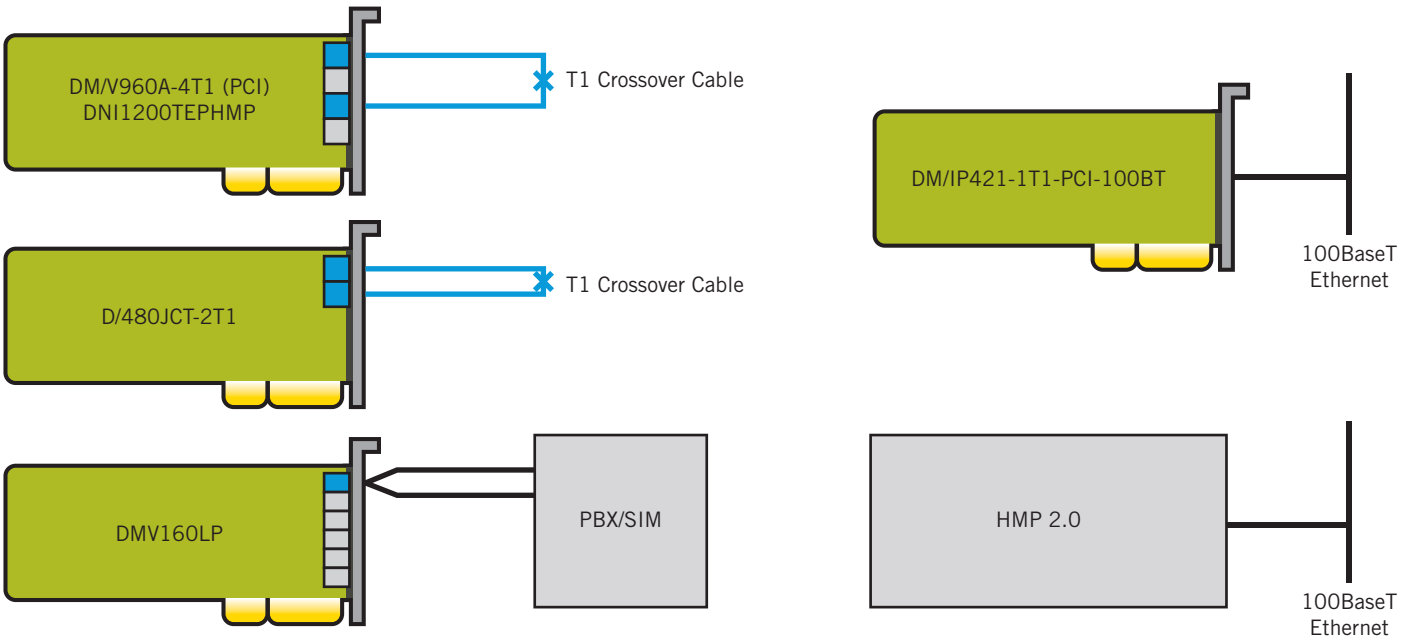

Figure 1. Sample Test Configuration

A T1/E1 crossover cable can be made by wiring 2 RJ-48C connectors in the following manner:

(Receive Ring)	1----------4	(Transmit Ring)
(Receive Tip)	2----------5	(Transmit Tip)
(Transmit Ring)	4----------1	(Receive Ring)
(Transmit Tip)	5----------2	(Receive Tip)

Command Line Usage

The sample test application supports a single channel when configured from the command line, or will support multiple channels when configured with a configuration file. Command line parameters include:

config=xxx	use xxx configuration file (one line per device) default = do not use configuration file sr6callp.cfg is used when no command line parameters are present
net=xxx	xxx = network device name
vox=xxx	xxx = voice device name
media=xxx	xxx = media device name (DM/IP board / HMP only)
type=xxx	xxx = technology type (DNI boards=DM3): [Dm3Analog|Dm3T1em|Dm3E1Cas |Dm3T1Isdn|Dm3E1Isdn |JctAnalog|JctT1em|JctE1Cas |JctT1Isdn|JctE1Isdn |IPLink|HMP] (IPLink parm is used for DM/IP boards)
protocol=xxx	xxx = protocol
cpa=n	n = call progress enabled [0|1], default = 1
cpaByCall=n	n = cpa configuration by call; otherwise, config by channel [0|1], default=1
useDxDialCpa=n	n = use Voice API (dx_dial) CPA method [0|1], default=0; forced to 1 for Springware ISDN and DM/IP and HMP
defaultCpa=n	n = use default CPA from cdp/.config [0|1], default=0
mode=xxx	xxx = application mode [outbound|inbound], default=outbound
maxCalls=n	n = maximum calls to process, default=1
dropOnConnected=n	n = drop call when connected and/or CPA [0|1], default=1
playFilename=xxx	xxx = filename for play command, default=sample.vox
playList=xxx	xxx = file contains a list of files to play (one for each call), default=none; playFilename is ignored if playList is specified
playOnOffered=n	n = play file on offered event [0|1], default=0. Note: When testing pre-connect responses, the playOnOffered parameter should be set to 1, so the file is played before accepting and answering the call.
destAddr=aaaa	aaaa = destination address, default = '7001'
callDelay=n	n = delay in seconds for next outbound call, default=5

Examples

This section contains two command line examples with a scenario and sample code.

Example 1

Make three outbound calls with CPA on the first channel of the first span on a single JCT card system configured to use the US T1 PDK protocol. Drop each call when it is connected, and delay 2 seconds between calls. Here is the sample code:

```
sr6callp net=dtiB1T1 vox=dxxxB1C1
type=JctT1em protocol=pdk_us_mf_io
maxCalls=3 callDelay=2
```

Example 2

Answer 3 inbound calls on the first channel of the second span on a single JCT card system configured to use US T1 PDK protocol. Wait for the far end to drop the call, respond upon connection with the recording sample1.vox on the first call, sample2.vox on the second call, and sample3.vox on the third call. Here is the command line code:

```
sr6callp net=dtiB2T1 vox=dxxxB7C1
type=JctT1em protocol=pdk _ us _ mf _ io
mode=inbound maxCalls=3
playList=playlist.txt
```

The file playlist.txt contains the following three lines:

```
sample1.vox
sample2.vox
sample3.vox
```

Configuration File Usage

A configuration file will be used if it is specified on the command line or if no command line parameters are specified. The device configuration parameters used in the configuration file are the same as those used on the command line and are organized in the configuration file with the parameters for each device on a separate line. Any number of stimulus (outbound) and/or response (inbound) channels may be configured.

Sample Test Application Output

The sample test application logs messages to the console window that show application progress, results, and errors. The sample output below is for a JCT T1 ISDN call that is answered with an answering machine:

```
06/19 12:50:31.906 Config file:
06/19 12:50:31.906  mode:outbound
06/19 12:50:31.906  type:JctT1isdn
06/19 12:50:31.906  protocol:isdn
06/19 12:50:31.906  net:dtiB2T1
06/19 12:50:31.906  vox:dxxxB7C1
06/19 12:50:31.906  destaddr:5001
06/19 12:50:31.906  maxcalls:1
06/19 12:50:31.906 Device Configuration:
06/19 12:50:31.906  net:dtiB2T1
06/19 12:50:31.906  vox:dxxxB7C1
06/19 12:50:31.906  media:
06/19 12:50:31.906  type:JctT1isdn
06/19 12:50:31.906  protocol:isdn
06/19 12:50:31.906  cpa:1
06/19 12:50:31.906  mode:Outbound
06/19 12:50:31.906  maxCalls:1
06/19 12:50:31.906  playFileName:sample.vox
06/19 12:50:31.906  playList:
06/19 12:50:31.906  playOnOffered:0
06/19 12:50:31.906  dropOnPlayComplete:0
06/19 12:50:31.906  dropOnConnected:1
06/19 12:50:31.906  cpaByCall:1
06/19 12:50:31.906  useDxDialCpa:1
06/19 12:50:31.906  defaultCpa:0
06/19 12:50:31.906  destAddr:5001
06/19 12:50:31.906  callDelay:5
06/19 12:50:31.906 starting GlobalCall...
06/19 12:50:35.015 GlobalCall started
06/19 12:50:35.015 dtiB2T1 DeviceState:opening
06/19 12:50:35.062 dtiB2T1 gc _ OpenEx :P _ isdn:N _ dtiB2T1
```

```
06/19 12:50:35.062 dxxxB7C1 dx _ open
06/19 12:50:35.062 dtiB2T1 GCEV _ UNBLOCKED
06/19 12:50:35.062 dtiB2T1 GCEV _ OPENEX gcDevice=:2
06/19 12:50:35.062 dtiB2T1 gc _ GetResourceH(GC _ NETWORKDEVICE)
06/19 12:50:35.062 dtiB2T1 Network Device dtiB2T1:2
06/19 12:50:35.078 dtiB2T1 dt _ getxmitslot
06/19 12:50:35.078 dtiB2T1 Net SCbus Timeslot:52
06/19 12:50:35.078 dtiB2T1 Voice Device dxxxB7C1:3
06/19 12:50:35.078 dtiB2T1 dx _ getxmitslot
06/19 12:50:35.078 dtiB2T1 Vox SCbus Timeslot:48
06/19 12:50:35.078 dxxxB7C1 dx _ listen
06/19 12:50:35.078 dtiB2T1 dt _ listen
06/19 12:50:35.078 dxxxB7C1 dx _ deltones
06/19 12:50:35.109 dxxxB7C1 dx _ initcallp
06/19 12:50:35.109 dtiB2T1 DeviceState:opened waiting for unblocked
06/19 12:50:35.109 dtiB2T1 DeviceState:active
06/19 12:50:35.109 dtiB2T1 DeviceState:resetting line device
06/19 12:50:35.109 dtiB2T1 gc _ ResetLineDev
06/19 12:50:35.109 dtiB2T1 GCEV _ RESETLINEDEV
06/19 12:50:35.109 dtiB2T1 DeviceState:active
06/19 12:50:35.109 dtiB2T1 calls processed: 0 / 1
06/19 12:50:40.109 dtiB2T1 CallState:dialing
06/19 12:50:40.109 dtiB2T1 gc _ MakeCall: 5001
06/19 12:50:40.125 dtiB2T1 GCEV _ PROCEEDING
06/19 12:50:40.140 dtiB2T1 dx _ dial
06/19 12:50:40.140 dtiB2T1 CallState:call proceeding
06/19 12:50:40.140 dtiB2T1 GCEV _ ALERTING
06/19 12:50:40.140 dtiB2T1 CallState:call alerting
06/19 12:50:40.156 dtiB2T1 GCEV _ CONNECTED
06/19 12:50:40.156 dtiB2T1 gc _ GetCallInfo
06/19 12:50:40.156 dtiB2T1 ConnectType: GCCT _ NA
06/19 12:50:43.265 dxxxB7C1 TDX _ CALLP
06/19 12:50:43.265 dxxxB7C1 CPA Result = CR _ CNCT:CON _ PAMD
06/19 12:50:43.265 dtiB2T1 CallState:call connected
06/19 12:50:43.265 dtiB2T1 CallState:dropping call
06/19 12:50:43.265 dtiB2T1 gc _ DropCall
06/19 12:50:43.281 dtiB2T1 GCEV _ DROPCALL
06/19 12:50:43.281 dtiB2T1 CallState:call idle
06/19 12:50:43.281 dtiB2T1 CallState:releasing call
06/19 12:50:43.281 dtiB2T1 gc _ ReleaseCall
06/19 12:50:43.296 dtiB2T1 GCEV _ RELEASECALL
06/19 12:50:43.296 dtiB2T1 CallState:null
06/19 12:50:43.296 dtiB2T1 calls processed: 1 / 1
Received signal SIGINT
06/19 12:50:53.296 dtiB2T1 dx _ unlisten
06/19 12:50:53.296 dtiB2T1 dt _ unlisten
06/19 12:50:53.296 dxxxB7C1 dx _ close
06/19 12:50:53.359 dtiB2T1 gc _ Close
06/19 12:50:53.359 dtiB2T1 DeviceState:closed
06/19 12:50:53.359 stopping GlobalCall...
06/19 12:50:53.359 GlobalCall stopped
06/19 12:50:53.359 done. hit any key...
```

Acronyms

API	Application programming interface
CPA	Call progress analysis
GTD	Global tone definitions
HMP	Host media processing
PAMD	Positive answering machine detection
PDK	Protocol developers kit
PVD	Positive voice detection
SIT	Special information tone
SR	System release

www.dialogic.com

Dialogic Inc
1504 McCarthy Boulevard
Milpitas, California 95035-7405
USA

06/12 10117-02