Dialogic®
Making Innovation Thrive™

Application Note

Migrating Dialogic® NaturalAccess™ API Call Control Functions to the Dialogic® R4 API

Dialogic® AG 2000 Series Media Boards and Dialogic® CX Series Station Interface Boards

Migrating Dialogic® NaturalAccess™ API Call Control Functions to the Dialogic® R4 API

Dialogic® AG 2000 Series Media Boards and Dialogic® CX Series Station Interface Boards

Application Note

Executive Summary

This document provides guidelines for migrating call control functions in existing applications using the Dialogic® NaturalAccess™ APIs for Dialogic® AG 2000 Series Media Boards and Dialogic® CX Series Station Interface Boards to similar functions in Dialogic® boards that use the Dialogic® R4 APIs. After a brief survey of the R4 APIs, this application note supplies tables that map NaturalAccess API functions, events, and call states to R4 API functions, events, and call states.

Note: For specific information about migrating from one board series to another, contact your Dialogic sales representative who can put you in touch with a Dialogic migration specialist and provide information about specific functionality equivalents. Dialogic has a broad range of product options available to which you can migrate, and a Dialogic migration specialist can help you choose among them.

Migrating Dialogic® NaturalAccess™ API Call Control Functions to the Dialogic® R4 API

Dialogic® AG 2000 Series Media Boards and Dialogic® CX Series Station Interface Boards

Application Note

Table of Contents

Migrating Dialogic® NaturalAccess™ API Call Control Functions to the Dialogic® R4 API

Dialogic® AG 2000 Series Media Boards and Dialogic® CX Series Station Interface Boards

Application Note

Getting Started

To move applications currently using Dialogic® NaturalAccess™ APIs to applications using Dialogic® R4 APIs, you will need to be familiar with the R4 API libraries and the various options available to you. R4 APIs is an umbrella term that includes the Dialogic® Voice API, the Dialogic® Global Call (GC) API, and the Dialogic® Modular Station Interface (MSI) API.

The Voice API and the GC API are used for call control. The Voice API is also used to control media (play, record, digit processing), and the MSI API is used for station manipulation.

Board Types

This application note specifically addresses call control in the following Dialogic® boards:

- **From** — Dialogic® AG 2000 Series Media Boards and Dialogic® CX Series Station Interface Boards
- **To** — Dialogic® Media Boards and Station Interface Boards with Dialogic® Springware Architecture. These boards are also called Dialogic® JCT Media Boards, Dialogic® Digital Station Interface (DISI) Boards, and Dialogic® High Density Station Interface (HDSI) Boards.

For specific information about migrating from one Dialogic board series to another, contact your Dialogic sales representative, who can put you in touch with a Dialogic migration specialist and provide information about specific functionality equivalents. Dialogic has a broad range of product options available to which you can migrate, and a Dialogic migration specialist can help you choose among them.

Summary of Dialogic® R4 APIs

Table 1 provides short descriptions of the R4 APIs with which you need to be familiar to migrate from an application currently using NaturalAccess APIs.

R4 API	Description
Dialogic® Standard Runtime Library (SRL)	Retrieves and manages asynchronous events and timeouts.
Dialogic® Modular Station Interface (MSI) Library	Manipulates analog stations, including ringing phones with standard or distinctive rings and detecting when a station goes on-hook, off-hook, or hook-flash.
Dialogic® Voice Library	A consolidated API that performs the basic steps to establish a call, such as setting on/off hook, dialing, call progress analysis, ring detection, and so forth. This API can be used for play and record processing. For call control, you can either use this API or the Dialogic GC API.
Dialogic® Global Call API Library	A high-level API designed so that a similar state machine/function set can be used to make calls on a variety of technologies. You can either use this API or the Voice API for call control.

Table 1. Dialogic® R4 APIs

API Documentation

Each of these APIs contains an associated programming guide and library reference:

- **Programming Guide** — contains many hints for application development and is useful to read before starting a detailed implementation.
- **Library Reference** — contains information about the functions, data structures and events used as part of the API library. It also shows sample code for each function.

You should familiarize yourself with these manuals before you begin planning your migration. They can be found at http://www.dialogic.com/manuals. Be sure to click the link for the specific Dialogic® System Release Software that you will be using.

Event Processing

The SRL allows you to wait, retrieve, and process asynchronous events, and it can retrieve and process application events for both the Voice API and the GC API.

The SRL offers several programming models, but the polled model is the model that is most similar to the NaturalAccess API. In the polled model, sr_waitevt() similar to the ctaWaitEvent() in the NaturalAccess API — these functions wait for and retrieve events and timeouts, which the application then processes. Because the SRL creates the needed event queues at application startup, it is not necessary to call a function such as ctaCreateQueue().

During the migration process, you can use SRL sr_putevt() to place a user-defined event onto the SRL event queue, which can be useful for generating events to map into your existing architecture. For example, if an autonomous function in the Voice API or GC API generates an event in your current implementation, you can call this function and then call sr_putevt() to generate the correct event, so that your state machine can progress.

Table 2 shows the mapping between NaturalAccess API and SRL API functions:

NaturalAccess API	SRL API
ctaCreateQueue()	N/A
ctaWaitEvent()	sr_waitevt()
N/A	sr_putevt()

Table 2. Mapping of NaturalAccess API and SRL API

Call Control Alternatives

Two options are available for handling analog voice control: the Voice API and the GC API.

The Voice API is a consolidated API that will only perform the basic steps needed to establish an analog call (setting on/off hook, dialing, call progress analysis, ring detection, etc.). The GC API is a higher level API than the Voice API, and allows a similar machine and function set to be used to make calls with a variety of technologies. For this reason, the GC API may be easier for a programmer familiar with the NaturalAccess API to use because API calls and call states are roughly equivalent. Call control for both APIs are discussed below.

Migrating Dialogic® NaturalAccess™ API Call Control Functions to the Dialogic® R4 API

Dialogic® AG 2000 Series Media Boards and Dialogic® CX Series Station Interface Boards

Call Control with the Dialogic® Voice API

The Voice API provides a comprehensive set of features for building and controlling voice applications.

Device Initialization

In the Voice API, you open a device by calling dx_open(). Because there is no need to initialize the libraries or construct event queues in the Voice API, dx_open() replaces the following NaturalAccess API functions:

- ctaInitialize()
- ctaCreateQueue()
- ctaCreateContext()
- ctaOpenServices()
- nccStartProtocol()

When calling dx_open(), you must provide a device name.

The standard device naming convention is: dxxxB***n***C***n***, where B***n*** is board number and C***n*** is the Voice channel number. The available device pool is segmented to 4 port cards; so, for example, port 6 on a board would be named dxxxB2C2.

The dx_open() function returns a device handle that is used to reference the device on subsequent API calls.

Once the device is opened, call dx_setevtmsk() to enable the LCOFF events for loop current disconnection, and the RING events for incoming call notification. After calling dx_setevtmsk(), the application will receive a DX_CST event every time ether of these call status transitions takes place. For an example that shows how to differentiate between DX_CST events during event processing, refer to the section on dx_setevtmsk() in the *Dialogic® Voice API Library Reference*.

Inbound Calling

The application receives a DX_CST event when the port detects ringing on the line. The application then has to check the associated data to confirm that the event is a DE_RINGS event, which is translated in the same way as the NCCEVN_INCOMING_CALL event. The application can answer by calling dx_sethook(DX_OFFHOOK), which returns a TDX_SETHOOK event after the call is answered.

Outbound Calling

For outbound calling, the application must call dx_sethook(OFFHOOK) to seize the analog line. Then, the application can call dx_dial() to dial the DTMFs that initiate the call. Optionally, call progress analysis (CPA) can be used to detect the outcome of the call on the PSTN, recording whether the call was answered, busy, or not answered. If answered, CPA detects whether the call was answered by voice, fax, or answering machine.

To create a call with CPA, set the value of the ca_intflg field in the DX_CAP structure before calling dx_dial(). For more information, refer to the *Dialogic® Voice API Programming Guide*.

Disconnecting a Call

If the line has a disconnect supervision such as TONE or LC Drop, the application receives a DX_CST event with a value of DELCOFF when the far end drops the call. If this occurs, set the application to call dx_sethook(ONHOOK) to drop the call from the application end.

Function and Event Mapping

Table 3 shows the mapping between NaturalAccess API functions and Voice API functions:

NaturalAccess API Function	Voice API Function
ctaCreateContext	dx_open
ctaOpenServices	dx_open
nccStartProtocol	dx_open followed by dx_setevtmsk
nccPlaceCall	dx_sethook followed by dx_dial
nccAnswerCall	dx_sethook
nccAcceptCall	NA
nccDisconnectCall	dx_sethook
nccReleaseCall	NA

Table 3. Mapping of NaturalAccess API Functions and Voice API Functions

Table 4 shows the mapping between NaturalAccess API events and Voice API events:

Natural Access API Event	Voice API Event
NCCEVN_SEIZURE_DETECTED followed by NCCEVN_INCOMING_CALL	DX_CST (DE_RING)
NCCEVN_ANSWERING_CALL followed by NCCEVN_CALL_CONNECTED	TDX_SETHOOK
NCCEVN_REMOTE_ANSWERED	TDX_CALLP
NCCEVN_CALL_DISCONNECTED or NCCEVN_CALL_RELEASED	DX_CST(DE_LCOFF) followed by TDX_SETHOOK

Table 4. Mapping of NaturalAccess API Events and Voice API Events

Call Control with the Dialogic® Global Call API

The GC API is a higher level API than the Voice API. Because the GC API abstracts protocol-specific functions, a GC-based application need only handle call states, rather than manually walk though the protocol. For example, in the GC API, gc_MakeCall() seizes the line, waits for a dial tone, dials digits, and so forth.

Although more complicated than the Voice API, the GC API may be more accessible for programmers familiar with the NaturalAccess API, because both APIs are high-level APIs, and both have similar machine states and event schemes.

Table 5 shows the mapping between the NaturalAccess API functions and GC API functions:

NaturalAccess API Function	Global Call API Function
ctaCloseServices	gc_Detatch
ctaCreateContext, ctaOpenServices, and nccStartProtocol, in sequence	gc_Open or gc_OpenEx, followed by gc_WaitCall if inbound call detection is needed
ctaCreateQueue	gc_Start
ctaDestroyQueue	gc_Stop
ctaGetContextInfo	gc_GetUsrAttr
ctaGetParmByName	gc_GetParm
ctaGetText	gc_ErrorInfo
ctaGetTextEx	gc_ErrorValue
ctaInitialize followed by ctaCreateQueue	gc_Start
ctaOpenServices	gc_Attach or gc_AttachResource
ctaSetParmByName	gc_SetParm
ctaStartTrace	gc_StartTrace
ctaStopTrace	gc_StopTrace
ctaWaitEvent	gc_GetMetaEvent or gc_GetMetaEventEx
nccAcceptCall	gc_CallAck or gc_AcceptCall, depending on the protocol
nccAnswerCall	gc_CallAck or gc_AnswerCall, depending on the protocol
nccDisconnectCall	gc_DropCall
nccGetCallStatus	gc_CRN2LineDev, gc_GetANI, gc_GetCallState, or gc_GetDNIS
nccGetExtendedCallStatus	gc_GetSigInfo
nccGetLineStatus	gc_GetLineDevState or gc_GetXmitSlot
nccPlaceCall	gc_MakeCall
nccRejectCall	gc_CallAck
nccReleaseCall	gc_DropCall, gc_ReleaseCall, or gc_ReleaseCallEx
nccStartProtocol	gc_WaitCall
nccStopProtocol	gc_Close
nccStopProtocol, ctaCloseServices, and ctaDestroyContext, in sequence	gc_Close
nccStopProtocol followed by nccStartProtocol,	gc_ResetLineDev
no-op (if an internal timeout is reached)	gc_CallAck
oamBoardGetProduct followed by oamGetKeyword	gc_GetCTInfo
swiDisableOutput	gc_UnListen
swiMakeConnection	gc_Listen

Table 5. Mapping of NaturalAccess API Functions and Global Call API Functions

Table 6 shows the mapping between NaturalAccess API events and GC API events:

NaturalAccess API Event	Global Call API Event
ADIEVN_PROTOCOL_ERROR	GCDEV_ERROR or GCEV_TASKFAIL
CTAEVN_CLOSE_SERVICES_DONE with the value field equal to CTA_REASON_FINISHED	GCEV_DETACH
CTAEVN_CLOSE_SERVICES_DONE with the value field different from CTA_REASON_FINISHED, indicating the reason of the failure	GCEV_DETACH_FAIL
CTAEVN_OPEN_SERVICES_DONE with the value field equal to CTA_REASON_FINISHED	GCEV_OPENEX followed by GCEV_ATTACH
CTAEVN_OPEN_SERVICES_DONE with the value field different from CTA_REASON_FINISHED	GCEV_OPENEX followed by GCEV_OPENEXFAIL
NCCEVN_ACCEPTING_CALL	GCEV_ACCEPT
NCCEVN_ANSWERING_CALL and NCCEVN_CALL_CONNECTED	GCEV_ANSWERED
NCCEVN_CALL_DISCONNECTED with reason code in value field set to NCC_DIS_TIMEOUT or NCC_DIS_REMOTE_NOANSWER	GCEV_DISCONNECTED and GCEV_CALLSTATUS
NCCEVN_CALL_RELEASED	GCEV_RELEASECALL
NCCEVN_EXT_CALL_STATUS_UPDATE with the value field set to CALL_STATUS_UUI	GCEV_USRINFO
NCCEVN_PLACING_CALL	GCEV_SETUP_ACK
NCCEVN_PLACING_CALL followed by NCCEVN_CALL_PROCEEDING	GCEV_PROCEEDING
NCCEVN_PROTOCOL_ERROR	GCEV_ERROR or GCEV_TASKFAIL
NCCEVN_PROTOCOL_EVENT with the value field ISDN_PROGRESS	GCEV_PROGRESSING
NCCEVN_REMOTE_ALERTING	GCEV_ALERTING
NCCEVN_REMOTE_ANSWERED and NCEVN_CALL_CONNECTED, in sequence	GCEV_CONNECTED
NCCEVN_CALL_DISCONNECTED	GCEV_DROPCALL
NCCEVN_CALL_RELEASED	GCEV_DROPCALL
NCCEVN_SEIZURE_DETECTED, followed by NCCEVN_INCOMING_CALL	GCEV_OFFERED
NCCEVN_START_PROTOCOL_DONE with the value field CTA_REASON_FINISHED	GCEV_RESETLINEDEV
NCCEVN_START_PROTOCOL_DONE with the value field different from CTA_REASON_FINISHED	GCEV_RESTARTFAIL
NCCEVN_STOP_PROTOCOL_DONE	GCEV_RESETLINEDEV

Table 6. Mapping of NaturalAccess API Events and Global Call API Events

Table 7 shows the mapping between NaturalAccess API call states and GC API call states:

NaturalAccess API Call State	Global Call API Call State
NCC_CALLSTATE_ACCEPTING	GCST_ACCEPT
NCC_CALLSTATE_PROCEEDING	GCST_ALERTING
NCC_CALLSTATE_CONNECTED	GCST_CONNECTED
NCC_CALLSTATE_SEIZURE	GCST_DETECTED
NCC_CALLSTATE_OUTBOUND_INITIATED	GCST_DIALING
NCC_CALLSTATE_DISCONNECTED	GCST_DISCONNECTED
NCC_CALLSTATE_RECEIVING_DIGITS	GCST_GETMOREINFO
NCC_CALLSTATE_DISCONNECTED	GCST_IDLE
NCC_CALLSTATE_INVALID	GCST_NULL
NCC_CALLSTATE_INCOMING	GCST_OFFERED
NCC_CALLSTATE_PROCEEDING	GCST_PROCEEDING
NCC_CALLSTATE_PLACING	GCST_SENDMOREINFO

Table 7. Mapping of NaturalAccess API Call States and Global Call API Call States

Station Control

The MSI API controls analog station devices, including ringing phones and detecting when a station goes on-hook, off-hook, or hook-flash. The functions for controlling station devices in the NaturalAccess API and the MSI API are similar, although not all MSI functions are asynchronous. The ms_setstparm() function, for example, is an atomic function. If your state machine requires an event for a function that does not provide one, call sr_putevt() to generate one as needed.

Table 8 shows the mapping between NaturalAccess API functions and MSI API functions:

NaturalAccess API Functions	MS API Function
cdiDisableBattery	ms_setstparm
cdiEnableBattery	ms_setstparm
cdiStartDTMFDetector	dx_getdig
cdiStartRing	ms_genring
cdiStopRing	ms_stopfn
cdiUserLed	ms_SetMsgWaitInd

Table 8. Mapping of NaturalAccess API Functions and MSI API Functions

Dialogic®
Making Innovation Thrive™

www.dialogic.com

Dialogic Corporation
9800 Cavendish Blvd., 5th floor
Montreal, Quebec
CANADA H4M 2V9

06/09 11490-01