CT-Connect

Programming Guide

Order Number: 05-0417-004

Revision/Update Information: This manual is an update

Software/Version: CT-Connect™ Server Version 3.0
CT-Connect Application Programming Interface Version 3.0

Publication date: June 9, 1998

For **Sales Offices** and other contact information, visit our website at **http://www.dialogic.com.**

Contents

2 CTC Routine Specifications

C Features Specific to the Lucent DEFINITY Generic

Index

Tables

About This Manual

This manual contains detailed descriptions of CT-Connect (CTC) Application Programming Interface (API) routines. It provides guidelines for using these routines and includes details of the operational differences for specific switches and CTC client platforms.

Audience

This manual is for programmers writing applications that use a link between a CTC server (a system running the CTC Server software) and a switch to provide users at client systems with computer-integrated telephony facilities.

It assumes that programmers are familiar with:

- Writing programs in Visual Basic™ or C. All CTC routines described in this manual are shown in C format. However, language-specific definitions files are provided in both C and Visual Basic.
- The CTC concepts described in the *CT-Connect Introduction*.
- Compiling and linking programs on the appropriate CTC client operating system.

Associated Documentation

CT-Connect Documentation

In addition to this manual, the following documents are included in the CTC documentation set:

- *CT-Connect Introduction* — This manual provides an overview of CTC and includes example configurations.
- *CT-Connect Installation and Administration Guide* for your CTC server platform — This manual describes how to install the CTC Server and the CTC API software on supported platforms. It also describes administration tasks and provides basic problem solving procedures.

- *CT-Connect Release Notes* — These online notes provide information about changes to the CTC software and/or documentation at the time of release. They are installed on the CTC server. For details of their location, refer to the *CT-Connect Installation and Administration Guide* for your CTC server platform.

Dialogic Web Site

For more information about CT-Connect, and other Dialogic products, visit Dialogic's web site at http://www.dialogic.com.

Switch Documentation

Refer to the documentation supplied with the switch for details of features, and any limitations that may affect the operation of the CTC software.

Terms and Definitions

The following terms are used throughout this manual:

Term	Definition
Windows 3.1/3.11	Refers to Microsoft® Windows™ 3.1, Windows 3.11, and Windows for Workgroups 3.11.
OpenVMS	Refers to the OpenVMS™ VAX™ and OpenVMS Alpha operating systems.
OS/2	Refers to the OS/2 Warp™ operating system.
CTC client	A supported system running the CTC API software.
CTC server	A supported system running the CTC Server software.
Communications link	The logical link between the CTC server and the switch.
Switch	The telephony switching device. For example, a Private Branch Exchange (PBX), Private Automatic Branch Exchange (PABX) or central office switch.

Conventions

The following conventions are used throughout this manual:

Convention	Meaning
courier	This typeface is used for code examples or interactive examples to indicate system input/output.
drive:	Italic (slanted) typeface indicates variable values, placeholders, and arguments.)
C:\>	The MS-DOS® and OS/2™ command prompt. The actual prompt may vary depending on your current drive and default directory.
#	The Digital™ UNIX®, HP-UX®, SCO OpenServer™, SCO® UnixWare®, and Sun™ Solaris™ command prompt.
$	The OpenVMS command prompt.

Part I

Part I describes how to use CTC programming routines and gives detailed descriptions of the routines.

1

Introduction

This chapter provides an overview of the CT-Connect (CTC) Application Programming Interface (API) routines. It describes the mechanisms and data structures you use to call CTC routines, describes how to use multithreaded programming with CTC, and provides guidelines for linking your programs.

1.1 CTC API Routines

CTC is a software toolkit for developing and running telephony applications. This section summarizes the functions that are available through CTC. For detailed information about the individual routines, refer to Chapter 2.

There are three groups of routines:

- Routines that control the communications channel
- Routines that perform telephony functions
- Switch-specific routines

1.1.1 Routines That Control the Communications Channel

The group of routines listed in Table 1–1 give you control of the communications channel between the user's application and a specific telephony device. Using the routines, you can:

- Assign and deassign logical communications channels to and from devices.
- Set and query certain characteristics for a device.
- Monitor telephony events for the device.

Monitoring Events

To receive event information for a device, you monitor the assigned channel to the device. Event information includes details of the current state of the device every time a significant event occurs on a channel.

If the current state of a device is known, the telephony features available at any one time can be predicted. For example, if there is a call active on a device, then you can present the user with the option to transfer the call.

Status messages consist of a combination of the following information:

- Current state
- Most recent event
- Identity of other parties
- Network information, for example, Dialed Number Identification Service (DNIS) or Automatic Number Identification (ANI)

This information allows you to build up and maintain the context for a sequence of calls. For example, you can keep track of a number of calls associated with a monitored device as the user swaps between active calls and calls on hold, or perhaps transfers a call on hold.

You can also receive event information on a monitor channel. A monitor channel is a single, logical channel that your application can create to monitor, but not control, multiple devices. By assigning to a monitor channel, your application can receive on one channel, call data, status, and party information for a number of devices. Your application can use this information, for example, to record statistics for a specific agent group.

Table 1–1 Controlling the Communications Channel

Function	Routine
Assign a communications channel to a device, and identify the channel uniquely to the application. The device can be a telephone, or a logical entity such as an ACD queue.	ctcAssign
Assign a monitor channel so that you can monitor a number of devices on a single channel.	ctcAssign
Receive event information for a device on a monitor channel.	ctcAddMonitor
Stop monitoring a device on a monitor channel.	ctcRemoveMonitor
Deassign a channel from its associated device, and release resources associated with the channel.	ctcDeassign
Return information about the communications channel and the device to which the channel is assigned.	ctcGetChannelInformation

Table 1–1 Controlling the Communications Channel (Continued)

Function	Routine
Return the number of calls at the device or in a queue, and query the state of those calls.	ctcSnapshot
Set status for an ACD agent so that they can log on or log off as an ACD agent and set the agent mode (for example, “ready to take calls”).	ctcSetAgentStatus
Set forwarding on for a device so that incoming calls are redirected to another device.	ctcSetCallForward
Set Do-Not-Disturb for a device so that incoming calls do not ring at the device.	ctcSetDoNotDisturb
Set the message waiting indicator on or off.	ctcSetMessageWaiting
Set the monitoring state of the assigned device on or off. Use this routine with the ctcGetEvent or ctcWinGetEvent routine to receive information on the state of calls associated with a device.	ctcSetMonitor
Enable or disable routing for the assigned route point. When routing is enabled, the switch passes route requests to CTC for incoming calls made to the assigned route point. The application can receive the route requests by using ctcGetRouteQuery or ctcWinGetRouteQuery, and can use ctcRespondToRouteQuery to specify a new destination for the incoming call.	ctcSetRoutingEnable
Show whether the switch passes route requests to CTC when a call reaches the assigned route point.	ctcGetRoutingEnable
Return current information on the status for an agent.	ctcGetAgentStatus
Return current information about call forwarding.	ctcGetCallForward
Return current information about the Do-Not-Disturb status.	ctcGetDoNotDisturb
Return the status of the message waiting indicator.	ctcGetMessageWaiting
Return information about the current monitoring state of the assigned device.	ctcGetMonitor
Return details of a condition value in text.	ctcErrMsg

Table 1–1 Controlling the Communications Channel (Continued)

Function	Routine
Return information on telephone calls associated with the assigned device: • Call states, such as initiate or active • Call events, such as answered or transferred • Call references (identifiers for calls) • Other parties involved in the telephone call, and network information such as ANI or DNIS	ctcGetEvent or ctcWinGetEvent
Associate data with a call (for example, customer reference information)	ctcAssociateData

Refer to Chapter 2 for detailed information on these routines.

1.1.2 Routines for Telephony Functions

Table 1–2 lists the telephony functions provided by the CTC API on a channel assigned to a device, and the routines that perform those functions.

Table 1–2 Telephony Functions

Telephony Function	Routine
Make a telephone call from the device to which the channel is assigned.	ctcMakeCall
Answer an incoming call on a hands-free feature telephone.	ctcAnswerCall
Pick up a call from another extension.	ctcPickupCall
Clear the active call on the assigned device.	ctcHangupCall
Put the current call on consultation hold.	ctcHoldCall
Make a call to a third party to whom you intend to transfer the current call on the assigned device, or to include all parties in a conference call.	ctcConsultationCall
Complete a transfer call, and disconnect the assigned device.	ctcTransferCall
Make a call and transfer the call without placing the calling party on hold (unsupervised transfer).	ctcSingleStepTransfer

Table 1–2 Telephony Functions (Continued)

Telephony Function	Routine
Merge two or more calls into a single conference call.	ctcConferenceJoin
Disconnect a consultation call.	ctcCancelCall
Retrieve a call that is on consultation hold.	ctcRetrieveHeld
Disconnect a consultation call and retrieve the held call.	ctcReconnectHeld
Swap the active call with the call on consultation hold.	ctcSwapWithHeld
Deflect a call ringing on the assigned device to another extension.	ctcDeflectCall
Notify a busy destination of the presence of your call, so that when the destination device finishes its current call, and your call is first in the queue, you are automatically connected. This is called camping on.	ctcRespondToInactiveCall
Barge in (also called intrude) on an existing call.	ctcRespondToInactiveCall
Get the switch to ring the assigned device back as soon as an extension that was previously busy becomes free.	ctcRespondToInactiveCall
Present a call to the call-center application so that it can decide to which device the call needs to be routed.	ctcGetRouteQuery or ctcWinGetRouteQuery
Route the incoming call to a destination chosen by the application.	ctcRespondToRouteQuery
Allow a virtual party on a switch to initiate calls on behalf of a user. Only when the called device answers, (or, for example, the telephone rings a preconfigured number of times) does the call get put through to the user.	ctcMakePredictiveCall[1]
Send DTMF (Dual-Tone Multi-Frequency) digits to simulate a user pressing keys on a touch-tone telephone.	ctcSendDTMF

[1]This may require external tone detection devices

1.1.3 Switch-Specific Routines

In addition to the standard CTC routines listed in Table 1–1 and Table 1–2, CTC provides some switch-specific routines.

These additional routines are provided as extensions to the CTC API and they enable your application to access features that are specific to a particular switch.

For example, your application can call the ctcMlpPlayMessage routine which is provided as a CTC API extension for the Nortel™ Meridian™ switches. This routine plays a voice message on a Meridian Mail system (see Section D.16 for more information).

You specify whether you want to use switch-specific routines when you assign a channel with the ctcAssign routine. See the description of ctcAssign in Chapter 2 for more information.

For this version of CTC, switch-specific routines are available for:

- Switches supporting CSTA Phase I and Phase II (see Appendix B)
- Lucent DEFINITY® G3 (see Appendix C)
- Nortel Meridian switches (see Appendix D)

For more information about future CTC API extensions, contact Dialogic.

1.2 Sequence for Calling CTC API Routines

To establish and control a logical channel to a device, and to receive information about activity on that device, call CTC routines in the following sequence:

1. ctcAssign to assign the channel to the device.
2. Routines that set characteristics for the device, such as ctcSetAgentStatus or ctcSetCallForward.
3. ctcSetMonitor routine to set monitoring on.
4. ctcGetEvent or ctcWinGetEvent routine to monitor the channel and device while the application is making and receiving telephone calls.

Following this sequence, you can call any other CTC API routines.

If a routine is not successful, you can use ctcErrMsg to interpret the returned value. Refer to Chapter 2 for more information.

At the end of a user’s session, when they have finished using the CTC application, ctcDeassign must be used to deassign the channel from the device.

1.3 Format of Routines

Chapter 2 describes each routine in detail, including:

- The format of the routine (written in C)
- A summary of the arguments passed to the routine

Arguments passed to a routine must be listed in your program in the same order as that shown in the format section.

1.3.1 Unsigned Integers and Windows 3.1/3.11

With the exception of ctcErrMsg, the format section for each routine in Chapter 2 shows status returns as 32-bit unsigned integers. On Windows 3.1/3.11, this is the equivalent of an unsigned longword.

If you are writing a Windows 3.1/3.11 program, use unsigned longwords wherever the format section or argument for a routine requires a 32-bit unsigned integer.

1.4 Use of Arguments

The Arguments section of a routine description describes the use of each argument. Each argument has three characteristics: data type, access type, and passing mechanism. For example, the channel argument has the following characteristics:

channel

type:	**ctcChanId**
access:	**read only**
mechanism:	**by value**

Section 1.4.1 to Section 1.4.3 describe these characteristics.

1.4.1 Data Type

When a calling program passes an argument to a CTC routine, the routine expects the argument to be of a particular data type. The *type* entry indicates the type of data used for an argument. CTC uses the following standard data types:

- Byte (unsigned) — 8 bits
- Word (unsigned) — 16 bits
- Integer (unsigned) — 32 bits (unsigned longword on Windows 3.1/3.11)
- Character string — Array of NUL-terminated bytes (signed)

CTC arguments can also be structures, unions, or arrays of these data types. These are described in Section 1.4.1.1 to Section 1.4.1.3. Section 1.4.1.4 summarizes the data, structures, unions, and arrays defined in CTC definitions files.

1.4.1.1 Data Structures

Some CTC arguments are addresses of data structures. A data structure is a block of memory that contains a series of fields of predefined offsets. Each of these structures has a fixed format that is defined in a CTC definitions file installed on your system (see Section 1.5, for more information about definitions files).

There are two types of structure:

- An input structure requires the application to pass information to the CTC API for one or more of the defined fields. For example, the ctcAssign routine requires the application to provide the dialable number (the directory number or extension number) of a telephone device. CTC has read-only access to the content of an input structure.
- Output structures are used to provide the application with information. The application program passes to CTC the address of a block of memory for the structure. CTC writes information into the structure for the application to read. CTC has write-only access to the content of an output structure.

  For example, CTC provides channel information in the ctcChanData structure for the ctcGetChannelInformation routine.

1.4.1.2 Unions

A union is an object that contains, at different times, any one of several elements of different types. For example, the privDataValue union contains an element that identifies the type of data passed with the ctcCstaEscape routine. Refer to Appendix B for details of this routine.

1.4.1.3 Arrays

An array is a sequence of data elements. For example, the callData argument for the ctcSnapshot routine is an array of up to 32 structures. For more information, refer to the description of ctcSnapshot in Chapter 2.

1.4.1.4 CTC Data Type Definitions

Table 1–3 provides a summary of CTC-defined data types.

Table 1–3 CTC Data Types

Data Type	Description
ctcAccountInfo	An array of bytes used to provide account information associated with a call.
ctcApplString	A character string that contains application data, for example, customer reference information. The maximum length of the string is specified by the literal ctcAppDataLen defined in a CTC definitions file.
ctcAssignData	A structure used to pass information required to create a channel to a device.
ctcCallData	A structure used to pass a reference and a state for a call.
ctcChanData	A structure used to pass information about the assigned channel.
ctcChanId	A pointer to a fixed structure used to identify the assigned channel.
ctcDeviceString	A character string usually containing the number for a device, for example, a DN. The maximum length for ctcDeviceString is specified by the literal ctcMaxDnLen defined in a CTC definitions file.
ctcEventData	A structure used to pass information relating to an event on the assigned channel.
ctcLogIdString	A null-terminated character string containing the logical identifier for the link between the CTC server and the switch.
ctcLpvASB	A Windows 3.1/3.11 structure containing information relating to an event on the assigned channel.
ctcNameString	A null-terminated character string containing the network name or address for the CTC server. The maximum length for ctcNameString is specified by the literal ctcNodeNameLen defined in a CTC definitions file.
ctcNetString	A null-terminated character string that identifies the network protocol used between the CTC client and CTC server. The maximum length for ctcNetString is specified by the literal ctcNetLen defined in a CTC definitions file.
ctcRouteData	A structure containing information relating to a call presented to the application for routing.

Table 1–3 CTC Data Types (Continued)

Data Type	Description
ctcTimeStamp	A structure containing details of the time an event or route request occurred.

Additional data types are used for switch-specific routines. For details of these data types, refer to the switch-specific appendixes.

1.4.2 Access to Data

The *access* entry indicates whether CTC:

- Reads data passed to it by the application (**read only**)
- Returns data to the application (**write only**)
- Reads data from the application and returns data to the application (**read and write**)

1.4.3 Passing Mechanism

The *mechanism* entry indicates whether the application passes data to the CTC routine by reference or by value:

- **By Value**

  When your program passes an argument by value, the argument entry contains the actual, uninterpreted value of the argument. The by value mechanism is usually used to pass constants. For example, to pass the constant 100 by value, the calling program puts 100 directly into the argument list.

- **By Reference**

  When your program passes an argument by reference, the argument entry contains the address of the location that contains the value of the argument. For example, if variable *x* is allocated at location 2000, which currently contains the value 100, the argument entry will contain 2000.

1.4.4 Passing Optional Data

For some arguments, passing data is optional. This means that you must still include the argument in your program but, depending on the passing mechanism, you can specify the value zero (0) or the address of a zero-length character string with the argument instead of data.

For example, the calledNumber argument for ctcPickupCall is the address of a character string that identifies the ringing device. You can specify the address of a zero-length character string with this argument to indicate that the call is being picked up from the local group.

The passing mechanism for the argument determines what you specify instead of the described data:

- If the argument is passed by value, specify zero (0) instead of the described value.
- If the argument is passed by reference and provides input to CTC, specify the address of a null data type, for example, a zero-length character string.
- If the argument is passed by reference and obtains output from CTC, supply enough memory to accommodate that argument's output.

To find out if you need to pass data with an argument, refer to the routine descriptions in Chapter 2.

1.5 Definitions

CTC supplies language-specific definitions files for constants, condition values, and data structures:

- If you are writing an application in C, you can include one definitions file, CTCDEF.H, in your program. This file includes files CTC_ERR.H, CTC_CODE.H, and CTC_RPC.H or CTCWIN16.H. Table 1–4 shows the contents of these files.

Table 1–4 C Definitions Files

File	Client Platforms
Constants	
CTC_CODE.H	All
Condition Values for Status Returns	
CTC_ERR.H	All
Data Structures	
CTC_RPC.H	Digital UNIX, HP-UX, OpenVMS, OS/2, SCO OpenServer, SCO UnixWare, Solaris, Windows 95, Windows NT™
CTCWIN16.H	Windows 3.1/3.11

- If you are writing an application in Visual Basic, you can include one definitions file, CTCDEF.BAS, in your program. This file includes all CTC definitions for Windows 3.1/3.11, Windows 95, and Windows NT.

Definitions files are copied to the directory that you specify during installation. For details of the location of the files, refer to the *CT-Connect Installation and Administration Guide* for your CTC server platform.

1.6 Condition Values for Status Returns

Each CTC routine returns a condition value (32-bit unsigned integer) as a completion code to indicate whether the call to the routine has been successful or whether an error has occurred.

Dialogic recommends that you always check the return status to determine success or failure of calls to CTC routines, and choose a suitable recovery path if there is an error.

For an explanatory list of the condition values that can be returned by CTC routines, refer to Chapter 3. Future versions of CTC may include additional values, so you should ensure that your application can handle future additions to the condition values.

1.6.1 Link Problems

If the link to the switch has gone down or reset, the CTC server:

- Clears all monitors and cancels any outstanding ctcGetEvent, ctcWinGetEvent, ctcGetRouteQuery, and ctcWinGetRouteQuery requests, returning a status of ctcLinkDown or ctcLinkReset.
- Returns a ctcLinkDown or ctcLinkReset condition value for outstanding or new API function calls until the link is re-enabled.

1.7 Exception Handling

A network problem that affects communication between the CTC client and CTC server may result in a Remote Procedure Call (RPC) exception.

The CTC API does not handle RPC exceptions automatically, so, Dialogic recommends that your application handles this type of exception. For details of how to do this, refer to the RPC programming documentation for your operating system.

1.8 Calling CTC Routines

All CTC routines operate synchronously. This means that they return to the caller only when the operation is complete.

Waiting for each operation to complete may be inappropriate for your application. For example, your application can use the ctcGetEvent routine to return information on telephone calls associated with the assigned device. This routine does not complete until there is activity on the assigned device. For your application to continue with operations, you must call CTC routines in a multithreaded program.

Multithreaded programming enables routines to be processed concurrently rather than in sequence. This means that applications are not blocked as they wait for operations to complete; operations that are asynchronous in nature can be performed in parallel with operations that are synchronous.

1.9 CTC and Multithreaded Programming

The following sections provide an overview of threads and multithreaded programs for applications that require both synchronous and asynchronous operations.

Note that you do not need to create a multithreaded program if:

- Your application uses only synchronous operations.
- You are writing a Windows 3.1/3.11 application. ctcWinGetEvent and ctcWinGetRouteQuery are non-blocking routines that allow a Windows 3.1/3.11 application to receive information for the assigned device. See Section 1.10 for more information.

1.9.1 Threads

A thread is a separate, sequential flow of control within a program. It is the movement of a processor through a program’s instructions.

1.9.2 Multithreaded Programming

Most traditional programs consist of a single thread. In a multithreaded program, multiple threads are created to execute different parts of a program. This enables a program to overlap activities.

Threads in a multithreaded program share the address space, memory (except for stacks and register contents), and other resources provided by a single process. When the process is created, a single thread is created and used by the program. This is the main thread. From this thread, the program can create another thread, for example, for an operation that needs to wait for input from another device. It continues to perform more immediate work using the main thread.

If the program has a number of operations to perform, it can create additional threads from the main thread as they are required.

1.9.3 Thread Execution

A processor executes a thread until the thread has to wait, for example, for a resource to become available, or for synchronization with another thread. At this point, the processor starts to run another thread. The processor continues in this way, executing one thread and then another.

No complicated data-passing mechanisms are required for one thread to communicate with another thread. A thread writes its output to memory and another thread can read it as input. When one thread has completed a task, it uses an indication mechanism (for example, a condition variable) to let the other thread know that the input data is ready.

1.9.4 Using Multithreaded Programming with CTC

Using multithreaded programming, a CTC application can complete both of the following activities:

- It can use the main thread (the thread created at the same time as the process) for all synchronous operations. For example, calling the ctcMakeCall routine.
- It can create another thread for monitoring the device. This operation is asynchronous in nature because the application waits for activity on the assigned device.

There are two routines that return information only when there is call activity, ctcGetEvent and ctcGetRouteQuery. Dialogic recommends that you create a separate thread for each of these routines if you use them in your program.

The online CTC application shows how multithreaded programming is used. This example application is installed as part of the CTC client software kit. For details of the location of the example application, refer to the *CT-Connect Installation and Administration Guide* for your CTC server platform.

1.9.5 Creating a Multithreaded Program

The procedure for creating threads in your program depends on the operating system you are using. For some operating systems, you may need to obtain a threads package.

For information about creating and using threads, refer to the application development documentation for your operating system:

- On Windows NT or Windows 95 systems, refer to the development documentation provided with your system. Threads are provided as part of the operating system for these platforms.
- On SCO UnixWare and SCO OpenServer systems, you can use DCE Thread Library routines. For more information, refer to the documentation provided with the SCO DCE Executive and SCO DCE Development System software.
- On Digital UNIX and OpenVMS systems, you can use the DCE Thread Library routines. These routines are described in the *Digital DCE Application Development Reference* manual.
- On HP-UX systems, you can use DCE Thread Library routines. For more information, refer to the documentation provided with the HP® DCE Runtime Services software.
- On Solaris systems, for details of DCE threads, refer to the documentation provided with the Transarc® DCE Version 2.0 for Solaris 2.6 software.
- On OS/2 systems, you can use DCE Thread Library routines. For more information, refer to the *IBM Distributed Computing Environment 2.1 for OS/2 Warp: Application Development Guide*. This guide is supplied online as part of the DCE Application Program Development Toolkit. The toolkit is available with the IBM® Directory and Security Server for OS/2 Warp software.

1.10 Using the CTC Windows Socket Interface

The CTC Windows Socket interface is installed with the CTC API on systems running Windows 3.1/3.11. It enables CTC applications running on these systems to use ctcWinGetEvent and ctcWinGetRouteQuery. These are non-blocking routines that return information for the assigned device. For more information, refer to the description of these routines in Chapter 2.

Note that ctcWinGetEvent and ctcWinGetRouteQuery are available on systems running Windows 3.1/3.11 only. If you are writing a Windows NT or Windows 95 application, you must use ctcGetEvent and ctcGetRouteQuery to receive event and routing information.

1.11 Example Programs

The following examples are installed during the CTC API installation procedure:

Example	Description
CTC_EXP.C	This file shows how to use the ctcAssign, ctcSetMonitor, and ctcGetEvent routines. It is available on all supported client platforms except Windows 3.1/3.11.
CTC Demo	This example application is installed on Windows NT and Windows 95 clients. The CTC API installation procedure installs source files in addition to the executable file.
Phone Watch	This example application is installed on Windows 3.1/3.11 clients. The CTC API installation procedure installs source files in addition to the executable file.

For details of the location of these files, refer to the *CT-Connect Installation and Administration Guide* for your CTC server platform.

1.12 Compiling and Linking Your Program

Section 1.12.1 to Section 1.12.9 contain platform-specific information about compiling and linking your program.

1.12.1 Digital UNIX Client

On an Digital UNIX client, the CTC client is provided as the shareable object, /usr/shlib/libctc_api.so. You include this shareable object as input when you link your program to create an executable image.

To compile your program, you use the cc -c command. For example:

```
# cc -c ctc_prog.c
```

where *ctc_prog.c* is source code written in C.

To link your program, you use the cc command and -l to specify the shareable object, dce, pthreads, c_r, and mach objects. For example:

```
# cc -o ctc_prog ctc_prog.o -lctc_api -ldce -lpthreads -lc_r -lmach
```

where *ctc_prog.o* is the compiled program and *ctc_prog* is the executable image.

1.12.2 HP-UX Client

On a CTC client running HP-UX, the CTC API is provided as the shareable library, libctc_api.sl. You include this shareable library as input when you link your program to create an executable image.

For example, to compile a program written in C, you use:

```
# cc -c -o ctcprog.o -Ae +O4 -I/usr/include/reentrant \
-D_REENTRANT ctcprog.c
```

where *ctcprog.o* is the name you give to the output (the compiled program) and *ctcprog.c* is source code written in C.

To link a program written in C, you use:

```
# ld ctcprog.o /lib/crt0.o -o ctcapp -s -Bimmediate -Bnonfatal \
-lctc_api -lbb -ldce -lm -lc_r
```

where *ctcprog.o* is the compiled program and *ctcapp* is the executable image.

1.12.3 OpenVMS Client

On an OpenVMS client, the CTC client is provided as the shareable image, SYS$SHARE:CTC_API.EXE. You include this shareable image as input to the linker.

Compile your program in the usual way and then complete the following procedure to link your image:

1. Create an options file that contains the following:

   ```
   SYS$SHARE:CTC_API.EXE/SHAREABLE
   ```

   Depending on the language you are using, you may also need to identify the Run-Time Library shareable image in the options file. For example, you create the following options file for a program written in C:

   ```
   SYS$SHARE:CTC_API.EXE/SHAREABLE
   SYS$SHARE:VAXCRTL.EXE/SHAREABLE
   ```

   where VAXCRTL.EXE is the shareable image for the VAX C Run-Time Library.

   For more information, refer to your language-specific programming utilities documentation.

2. Use the LINK command to link your image:

   ```
   $ LINK ctc_program, filename.OPT/OPTION, DCE:DCE.OPT/OPT ION
   ```

   where *ctc_program* is your compiled program and *filename*.OPT is the name of your options file. DCE.OPT is the DCE options file.

1.12.4 OS/2 Client

The following is an example command use to compile a program on a CTC client running OS/2:

```
icc ctprog.c /Gm+ /Su4 /Ms /C+ /Sem -D CMS_PROTO_ -D_CMA_NOWRAPPERS_
-DCMA_UNIPROCESSOR -DINTEL80x86 -IDBMOS2
```

where *ctprog.c* is source code written in C. This produces a compiled program ctprog.obj.

The following example shows how to link the CTC program:

```
ilink ctprog.obj /E /NOI /NOE /ST:100000 /O:ctprog ctapi.lib
dceos2.lib os2386.lib
```

where *ctprog.obj* is the compiled program and *ctprog* is the executable image.

1.12.5 SCO OpenServer Client

On a CTC client running SCO OpenServer, the CTC API is provided as the file libctc_api.a. You include this file when you link your program to create an executable image.

The following example shows how to compile a program on a CTC client running SCO OpenServer:

```
# cc -c -o ctcprog.o -belf ctcprog.c
```

where *ctcprog.o* is the name you give to the output (the compiled program) and *ctcprog.c* is source code written in C.

To link your program, you use:

```
# ld ctcprog.o /lib/crt0.o -o ctcapp -s -lctc_api -ldce -lcma -lm \
-lsocket -lc
```

where *ctcprog.o* is the compiled program and *ctcapp* is the executable image.

1.12.6 SCO UnixWare Client

On a CTC client running SCO UnixWare, the CTC API is provided as the shareable object, libctc_api.so. You include this shareable object as input when you link your program to create an executable image.

The following example shows how to compile a C program on SCO UnixWare:

```
# cc -c -Kpic, thread ctcprog.c
```

where *ctcprog.c* is CTC source code written in C.

The following example shows how to link the program:

```
# ld -s -Bdynamic -o ctcapp /usr/ccs/lib/crt0.o ctcprog.o \
-lctc_api -ldce -lc
```

where *ctcapp* is the executable image and *ctcprog.o* is the compiled program.

1.12.7 Solaris Client

On a CTC client running Solaris software, the CTC API is provided as the shareable object, libctc_api.so. You include this shareable object as input when you link your program to create an executable image.

The following example shows how to compile a C program on Solaris:

```
# cc -c -D_REENTRANT -Dsparc -Kpic ctcprog.c
```

where *ctcprog.c* is CTC source code written in C.

The following example shows how to link the program:

```
# ld -s -o ctcapp /opt/SUNWspro/SC3.0.1/lib/crt0.o ctcprog.o \
-lctc_api -ldce -lc -lnls -lthread -lm
```

where *ctcapp* is the executable image and *ctcprog.o* is the compiled program.

1.12.8 Windows 3.1/3.11 Client

Dialogic recommends that when you compile a CTC program on a system running Windows 3.1/3.11:

- You copy the following header files to your INCLUDE directory:

  CTCDEF.H
  CTC_CODE.H
  CTC_ERR.H
  CTCWIN16.H

- You use the large memory model.

You can link your program using one of the following methods:

- Implicit Import — This gives you access to all the CTC routines by including the import library in the linker command.
- Dynamic Run-Time Import — This allows access to only the routines you specify within your application code.

Implicit Import

To link your application with CTC, copy the CTC import library CTC_API.LIB to your library directory, and include it in the linker command file. For example:

```
link /NOD/CO ctcapp.obj,,ctcapp.map/map,libw llibcew
ctc_api.lib,ctcapp.def
```

Dynamic Run-Time Import

Dynamic Run-Time Import eliminates the need for you to link your program with the CTC import library. Your application first loads the CTC import library, and then retrieves the address of the CTC functions you specify.

For example, to call the ctcHangupCall routine:

```
HANDLE hLibrary;
FARPROC lpFunc;

hLibrary = LoadLibrary (“CTC_API.DLL”);
if (hLibrary >= 32)
     {
      lpFunc = GetProcAddress (hLibrary, “ctcHangupCall”);
      if (lpFunc !=(FARPROC)NULL)
            (*lpFunc) (hChan);
      FreeLibrary (hLibrary);
      };
```

1.12.9 Windows 95 and Windows NT Clients

This section contains information about compiling and linking your program on a Windows 95 or Windows NT system.

Calling Convention for Linking

CTC API functions conform to the **stdcall** calling convention for C and C++ programs. To link and run CTC applications, CTC provides stdcall-compatible files CTCAPI32.DLL and CTCAPI32.LIB.

For details of the location of these files, refer to the *CT-Connect Installation and Administration Guide*. For more information about the stdcall calling convention, refer to your C or C++ documentation.

Multithreaded Programs and Thread Stack Size

If you create threads in your program, note the following:

- On Windows 95 systems, if you encounter problems with virtual memory, try reducing the thread stack size when you link your program. For more information, refer to your Windows 95 documentation.
- On Windows NT systems, Dialogic recommends that you use a thread stack size of no more than 64 Kbytes when you link your program. The default thread stack size on Windows NT systems is 1 Mbyte.

Paths

During the CTC API installation, the file CTCVARS.BAT is copied to your PC. This file contains the following paths:

> *drive*:*directory*\\LIB
> *drive*:*directory*\\INCLUDE

where *drive*:*directory* is the drive and directory used for the CTC API installation. By default, this is C:\PROGRAM FILES\DIALOGIC\CTC API.

These paths define the location of the CTC API library and definitions files.

1.13 Changes to CTC for Version 3.0

Table 1–5 provides a summary of new features and routines provided with this version of CTC.

Table 1–5 Summary of Changes to CTC for V3.0

This version of CTC adds support for...	Change to CTC API...	For more information, refer to...
Private data for: • CSTA Phase I switches • Ericsson® MD110 (BC9) • Siemens® Hicom® 300E V3	New elements for the following routines: ctcCstaEscape, ctcCstaGetPrivateData, ctcCstaGetPrivateEventData, ctcCstaGetPrivateRouteData, ctcCstaSetPrivateData. Also: • ctcAssociateData is supported for the Ericsson MD110. • Application data can be passed with the ctcDeflectCall and ctcConsultationCall routines for the Hicom 300E V3.	The following: • For information about API extensions, Section 1.1.3 and the description of ctcAssign in Chapter 2. • For details of CSTA routines, Appendix B. • For details of the ctcAssociateData, ctcDeflectCall, and ctcConsultationCall routines, Chapter 2 and Appendix B.
Additional routine available for Lucent DEFINITY G3 switches.	The APIextensions field of the ctcAssignData structure supports an additional value, ctcK_ASAI. New routine available: ctcAsaiGetAcdStatus.	The following: • For information about API extensions, Section 1.1.3 and the description of ctcAssign in Chapter 2. • For details of the ctcAsaiGetAcdStatus routine, Appendix C.

Table 1–5 Summary of Changes to CTC for V3.0 (Continued)

This version of CTC adds support for...	Change to CTC API...	For more information, refer to...
Additional processing option for predictive calls available for Lucent DEFINITY G3 switches.	Additional value (ctcK_AllocAMDAdmin) supported for the ctcMakePredictiveCall allocation argument.	The description of ctcMakePredictiveCall in Chapter 2 and Lucent DEFINITY G3 support for the allocation argument in Appendix C.
Improved link failure detection, particularly for TCP/IP connections. The CTC server can periodically poll the link to the switch, checking whether it is still enabled. If the switch does not acknowledge two consecutive poll requests from the CTC server, the CTC server assumes that the link has failed and restarts it.	None.	The following: • For details of how the CTC server reports link problems to the CTC API, see Section 1.6.1. • For details of how to use the Configuration Program to enable link state checking, see the *CT-Connect Installation and Administration Guide* for your CTC server platform.

1.14 Compatibility With Previous Versions of CTC

If you are upgrading from CTC Version 2.0, note the following:

- CTC clients running Version 2.0 or 3.0 of the CTC API are compatible with a CTC server running Version V3.0 of the CTC Server software.
- You can continue to use CTC Version 2.0 applications on a client system running CTC API Version V3.0 software.

These temporary measures enable your CTC network and CTC applications to be upgraded progressively. For more information about installing the software, refer to the *CT-Connect Installation and Administration Guide* for your CTC server platform.

Note that:

- When you use ctcAssign, the APIversion value ctcK_CurrentVersion is the equivalent of ctcK_CTCV30 for this version of CTC. Programs compiled with either of these values will automatically access the new interface to modified CTC API functions. For more information about ctcAssign, refer to Chapter 2.

- The default calling convention for CTC V2.0 and V3.0 applications on Windows™ 95 and Windows NT™ is stdcall. For CTC API V1.1/1.11, the default convention was cdecl. If you recompile CTC V1.1/V1.11 applications on CTC clients running CTC API V3.0, you will need to modify your build procedure. Refer to Section 1.12.9 for details.

2

CTC Routine Specifications

This chapter gives detailed descriptions of CTC routines, in alphabetical order.

The descriptions indicate how to invoke telephony functions through CTC, but do not describe how the functions work on specific switches. Refer to the documentation supplied with your switch for details of how they work on that particular switch. For example, PBX documentation should indicate the maximum number of parties allowed on a conference call.

Because of the differences between protocols and switches, there may be differences in the routines and features made available over the CTC link. Appendix A tells you which CTC routines and features are common to all CTC protocol/switch links. The appendixes that follow tell you which routines and features are specific to the individual links supported by this release of CTC.

ctcAddMonitor
Adds a Device to a Monitor Channel

Format in C

unsigned int ***ctcAddMonitor*** (*ctcChanId monitorChannel,*
*ctcAssignData *assignData)*

Description

The ctcAddMonitor routine associates a device with a monitor channel and sets monitoring on for that device so that event information is returned on the monitor channel. A monitor channel is a single logical channel you use for monitoring multiple devices, such as, telephones and route points.

Only monitoring is supported on a monitor channel. If you want to use CTC routines to perform telephony functions for a device, you must use ctcAssign to assign a channel to the device.

Monitoring on a Monitor Channel

To set up a monitor channel, you use the following routines:

1. ctcAssign to create a monitor channel
2. ctcAddMonitor for each device you want to monitor on the monitor channel
3. ctcGetEvent to return information on the monitor channel

To identify which monitor channel is used to return information about a device, you specify the channel identifier with the monitorChannel argument. To identify the device, you specify its DN. Event information for the device is then returned on the monitor channel.

Monitoring Another Monitor Channel

To monitor another monitor channel, use the ctcGetChannelInformation to obtain a device number for the monitor channel (returned in the setDN field of the ctcChanData structure) and specify this as the deviceDN with the assignData argument.

Note that:

- You can only use one level of nested monitoring for monitor channels. This means that you cannot monitor a monitor channel if that channel is already monitoring another monitor channel.
- A monitor channel cannot monitor itself.

Removing Monitoring for a Device

To stop monitoring a device on the monitor channel, you use ctcRemoveMonitor. For more information, see the description of the ctcRemoveMonitor routine.

Restriction

This routine is not supported on CTC clients running Windows 3.1/3.11. CTC applications running on Windows 3.1/3.11 cannot assign to monitor channels.

Arguments

monitorChannel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the monitor channel.

You use this argument to identify the monitor channel that will be used for monitoring the device.

The ctcChanId datatype is defined in a CTC definitions file (see Section 1.5).

assignData

type: **ctcAssignData**
access: **read only**
mechanism: **by reference**

This argument contains the address of a fixed-format structure, for which you allocate memory of type ctcAssignData. The structure is defined in a CTC definitions file (see Section 1.5) and is formatted as follows:

```
   ctcAssignData  {
      unsigned short      deviceType;
      unsigned char       APIversion;
      unsigned char       APIextensions;
      ctcDeviceString     deviceDN;
 };
```

The following information is required in the ctcAssignData fields:

- **deviceType**

  This 16-bit field identifies the type of device you are assigning to the monitor channel. You can use the monitor channel to monitor the following:

  – Telephony devices, such as, telephones, multiline sets, groups (queues), or Voice Response Units (VRUs)

– Route points (logical devices used for call routing)

– Monitor channels

Specify one of the following values in the deviceType field:

Value	Description
ctcK_Dn	Specifies that a telephony device will be monitored
ctcK_RoutePoint	Specifies that a route point will be monitored
ctcK_MonitorChannel	Specifies that a monitor channel will be monitored

For details of support for additional deviceType values, refer to the switch-specific appendixes.

- **APIversion**

This 8-bit field identifies the version of CTC API used. This ensures compatibility with previous and new versions of the CTC software when you compile your application. Specify one of the following values:

Value	Description
ctcK_CTCV20	Specify this value if you have written a CTC application for use only with version 2.0 of the CTC API software.
ctcK_CTCV30	Specify this value if you are writing a CTC application for use only with Version 3.0 of the CTC API software.
ctcK_CurrentVersion	Specify this value and your application will be compatible with the current version of the CTC API installed on your CTC client system. When you upgrade to a future version of the CTC API, your application will automatically gain access to any new events provided as part of that CTC API.

Dialogic recommends you use ctcK_CurrentVersion to ensure future compatibility.

If an application passes the value ctcK_CTCV11 or null data in the APIversion field, it will not work with a CTC V3.0 server. CTC returns the condition value ctcUnsupAPIversion (1037).

- **APIextensions**

  Use this 8-bit field to indicate whether your application is portable and can be used with CTC links to various switches, or a switch-specific application that makes use of CTC API extensions. CTC API extensions are additional, switch-specific functions (for example, ctcMlpPlayMessage for Nortel Meridian switches). Specify one of the following values:

Value	Description
ctcK_None	This indicates that your application is portable for links to different CTC-supported switches, and will not support switch-specific extensions to the CTC API.
ctcK_CstaPrivate	This indicates that your application will use both standard CTC functions and additional CTC functions that are only available for CSTA switches. These functions are described in Appendix B.
ctcK_ASAI	This indicates that your application will use both standard CTC functions and CTC functions that are specific to Lucent DEFINITY switches. For more information about these switch-specific functions, see Appendix C.
ctcK_MeridianLink	This indicates that your application will use both standard CTC functions and CTC functions that are specific to Nortel Meridian switches. For more information about these switch-specific functions, see Appendix D.

- **deviceDN**

  This 24 byte field identifies the device to be monitored on the monitor channel:

  – For a telephony device or route point, use this field to specify its directory number (telephone number). This is an ASCII string that can contain any combination of numbers 0 through 9 and the characters * and #.

  – For a monitor channel, specify the device number returned in the setDN field of the ctcChanData structure. This is returned when you call ctcGetChannelInformation for the monitor channel. See the description of ctcGetChannelInformation for more information.

    Note that you must specify the device number exactly as it is returned in the setDN field, using the same case for letters. The device number is an ASCII string that can contain any combination of numbers 0 through 9, uppercase letters A through F, and the characters * and #.

  The maximum length for deviceDN is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

ctcAnswerCall
Answer a Call

Format in C

unsigned int ***ctcAnswerCall*** *(ctcChanId channel,*
unsigned int callRefId)

Description

Use the ctcAnswerCall routine whenever CTC notifies you there is an incoming call to the assigned device and the user wishes to answer the call.

This routine is useful for hands-free operation on a feature phone because the user can answer the telephone without lifting the handset. However, it cannot be used with standard telephones.

CTC notifies you of an incoming call only if both of the following conditions apply:

1. You have set monitoring on, using ctcSetMonitor.
2. You are using the ctcGetEvent or ctcWinGetEvent routine, which indicates a change in state to receive (ringing).

The call reference identifier for the incoming call is returned by ctcGetEvent or ctcWinGetEvent.

Arguments

channel

type:	**ctcChanId**
access:	**read only**
mechanism:	**by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

The ctcChanId datatype is defined in a CTC definitions file (see Section 1.5).

callRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the call reference identifier for the incoming call. Specify the call reference identifier returned by ctcGetEvent or ctcWinGetEvent for the incoming call.

ctcAssign
Assign a Channel

Format in C

*unsigned int **ctcAssign** (ctcChanId *channel,*
*ctcAssignData *assignData,*
ctcNameString serverName,
ctcLogIdString logicalIdentifier,
ctcNetString networkType)

Description

Before a device (for example, a telephone) can be linked to a CTC network, the device and the communications channel must be uniquely identified to CTC by your application.

The ctcAssign routine assigns a logical communications channel between the application, the CTC server system, and the specified telephone device, and then returns an identifier (ID) for that channel.

When to Use ctcAssign

You must use the ctcAssign routine before any of the other CTC routines so that you know the channel ID associated with a device. All subsequent CTC routines that you invoke for the device require you to specify that channel ID.

If you use another CTC routine before ctcAssign, an error message is returned. For example, on Windows NT, a system error 6 (ERROR_INVALID_HANDLE) or the CTC condition value ctcRpcConnecFail.

You need assign a channel only once for each user session; that is, you do not have to assign and deassign the channel for each telephone call a user makes from a particular phone.

Monitor Channels

If you need to monitor a number of devices, you can use ctcAssign to create a single monitor channel that receives events for all the devices. For example, your application can create one channel to receive event information for all devices in an office building or for a particular group.

To set up a monitor channel, you use the following sequence of routines:

1. ctcAssign to assign a monitor channel
2. ctcAddMonitor for each device you want to monitor on the monitor channel

3. ctcGetEvent to return information for all devices associated with the monitor channel

Note that you do not need to use ctcSetMonitor in this sequence; when you use ctcAddMonitor, monitoring is automatically enabled for the device.

Routines Supported for Monitor Channels

The following subset of CTC routines are supported for channels assigned to monitor channels:

ctcAddMonitor
ctcAssign
ctcDeassign
ctcErrMsg
ctcGetChannelInformation
ctcGetEvent
ctcRemoveMonitor

Restriction: Monitor Channels and Windows 3.1/3.11

Monitor channels are not supported on CTC clients running Windows 3.1/3.11. If you try to assign to a monitor channel from Windows 3.1/3.11, CTC returns an error message, for example, ctcInvDN.

Arguments

channel

type:	**ctcChanId**
access:	**write only**
mechanism:	**by reference**

The channel argument is a pointer to datatype ctcChanId which receives the identifier for the channel. The ctcChanId datatype is defined in a CTC definitions file (see Section 1.5).

The channel ID is the value used by the other CTC routines to identify the device or monitor channel used.

For an example of the ctcChanId argument and how to use ctcAssign, refer to the example file CTC_EXP.C installed on your CTC client (not available on Windows 3.1/3.11 clients). For location details, refer to the *CT-Connect Installation and Administration Guide* for your CTC server platform.

assignData

type: **ctcAssignData**
access: **read only**
mechanism: **by reference**

This argument contains the address of a fixed-format structure, for which you allocate memory of type ctcAssignData. The structure is defined in a CTC definitions file (see Section 1.5) and is formatted as follows:

```
   ctcAssignData  {
      unsigned short      deviceType;
      unsigned char       APIversion;
      unsigned char       APIextensions;
      ctcDeviceString     deviceDN;
 };
```

The following information is required in the ctcAssignData fields:

- **deviceType**

  This 16-bit field identifies the type of device to which you are assigning. You can assign a channel to the following:

  – A telephony device. For example, a telephone, multiline set, group (queue), ACD agent, or Voice Response Unit (VRU).

  – A route point. A route point is a logical device used for call routing. It has a dialable number but there is no real, physical device.

    When a call reaches the route point, the switch passes a request for the call's destination to your application. Your application can use the ctcGetRouteQuery and ctcRespondToRouteQuery routines to obtain information about the call and reroute it to another destination.

  – A monitor channel. This is a single channel you use to monitor multiple telephony devices, route points, and other monitor channels.

    Monitor channels are not supported on CTC clients running Windows 3.1/3.11.

  Additional device types may be supported by your switch. For details, refer to the switch-specific appendixes.

  To assign to a telephony device, route point, or monitor channel, specify one of the following values in the deviceType field:

Specify this value...	To assign to...
ctcK_Dn	A telephony device
ctcK_RoutePoint	A route point
ctcK_MonitorChannel	A monitor channel

- **APIversion**

This 8-bit field identifies the version of CTC API used. This ensures compatibility with previous and new versions of the CTC software when you compile your application. Specify one of the following values:

Value	Description
ctcK_CTCV20	Specify this value if you have written a CTC application for use only with Version 2.0 of the CTC API software.
ctcK_CTCV30	Specify this value if you are writing a CTC application for use only with Version 3.0 of the CTC API software.
ctcK_CurrentVersion	Specify this value and your application will be compatible with the current version of the CTC API installed on your CTC client system. When you upgrade to a future version of the CTC API, your application will automatically gain access to any new events provided as part of that CTC API.

Dialogic recommends you use ctcK_CurrentVersion to ensure future compatibility.

If an application passes the value ctcK_CTCV11 or null data in the APIversion field, it will not work with a CTC V3.0 server. CTC returns the condition value ctcUnsupAPIversion (1037).

- **APIextensions**

  Use this 8-bit field to indicate whether your application is portable and can be used with CTC links to various switches, or a switch-specific application that makes use of CTC API extensions. CTC API extensions are additional, switch-specific functions (for example, ctcMlpPlayMessage for Nortel Meridian switches). Specify one of the following values:

Value	Description
ctcK_None	This indicates that your application is portable for links to different CTC-supported switches, and will not support switch-specific extensions to the CTC API.
ctcK_CstaPrivate	This indicates that your application will use both standard CTC functions and additional CTC functions that are only available for CSTA switches. These functions are described in Appendix B.
ctcK_ASAI	This indicates that your application will use both standard CTC functions and CTC functions that are specific to Lucent DEFINITY switches. For more information about these switch-specific functions, see Appendix C.
ctcK_MeridianLink	This indicates that your application will use both standard CTC functions and CTC functions that are specific to Nortel Meridian switches. For more information about these switch-specific functions, see Appendix D.

- **deviceDN**

  If you are assigning a channel to a telephony device or route point, use this field to specify its directory number (telephone number). This is an ASCII string that can contain any combination of numbers 0 through 9 and the characters * and #. The maximum length for deviceDN is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

  If you are assigning to a monitor channel, specify a zero-length character string.

serverName

type: **ctcNameString**
access: **read only**
mechanism: **by reference**

This argument is the address of a character string that contains the network name or address for the CTC server.

The maximum length for serverName is specified by the literal ctcNodeNameLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

Note that if you specify an invalid network name or address on Windows 95 or Windows NT, system error 1722 (RPC error rpc_s_server_unavailable) is reported. If you use ctcErrMsg to map this error, it returns ctcServerUnknown.

logicalIdentifier

type: **ctcLogIdString**
access: **read only**
mechanism: **by reference**

This argument is the address of a character string that contains an identifier for the link. The identifier is assigned to the link at the CTC server and is defined either during the installation of the server software, or after the installation with the CTC Control Program (see the *CT-Connect Installation and Administration Guide* for your CTC server platform).

The maximum length for logicalIdentifier is specified by the literal ctcLogIdLen in the CTC definitions file. Note that this maximum length includes the null termination character (NUL).

networkType

type: **ctcNetString**
access: **read only**
mechanism: **by reference**

This argument is the address of a character string value that identifies the network protocol used between the CTC client and the CTC server.

The network protocol that you specify must be supported by the CTC server, CTC client, and RPC. On Windows 3.1/3.11 clients, only TCP/IP is supported. For other CTC client systems, check with the system manager of your CTC network for details.

Specify one of the values in the following table:

Network Protocol	Value
NetBIOS™ over NetBEUI	ncacn_nb_nb
TCP/IP	ncacn_ip_tcp
DECnet™	ncacn_dnet_nsp
NetBIOS over TCP/IP	ncacn_nb_tcp
Named pipes	ncacn_np
Novell® SPX	ncacn_spx

The maximum length for networkType is specified by the literal ctcNetLen in the CTC definitions file. Note that this maximum length includes the null termination character (NUL).

ctcAssociateData
Send Call Data to the Switch

Format in C

unsigned int ***ctcAssociateData*** *(ctcChanId channel,*
unsigned int callRefId,
ctcApplString applicationData)

Description

The ctcAssociateData routine enables you to associate data with a call and pass it to the switch. For example, you can use this routine to associate customer reference information or account details with a call.

The data is stored by the switch and reported on subsequent events until the call is terminated.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

callRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This argument is a 32-bit integer that contains the call reference identifier. This identifier is returned by the ctcGetEvent or ctcWinGetEvent routines, and routines such as ctcMakeCall.

applicationData

type: **ctcApplString**
access: **read only**
mechanism: **by value**

This argument is the address of a character string that contains the data to be associated with the call.

ctcAssociateData

The maximum length for applicationData is specified by the literal ctcAppDataLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

ctcCancelCall
Cancel a Consultation Call

Format in C

unsigned int ***ctcCancelCall*** *(ctcChanId channel,*
unsigned int callRefId)

Description

If you make a consultation call, placing the original call on hold, you can cancel or hang up the consultation call with ctcCancelCall. This routine disconnects the consultation call (returning the assigned device to the initiate state), at which point you can either use ctcConsultationCall to make another call or ctcRetrieveHeld to return to the first call. Always use this routine to cancel or hang up a consultation call rather than ctcHangupCall because, on some switches, ctcHangupCall transfers the original call.

You can also use ctcReconnectHeld to cancel a consultation call and return to the first call. ctcReconnectHeld has the same effect as using both ctcCancelCall and ctcRetrieveHeld.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

callRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the call reference identifier for the consultation call to be cancelled. Specify the call reference identifier returned by ctcGetEvent or ctcWinGetEvent for the consultation call or other routines such as ctcConsultationCall.

ctcConferenceJoin
Merge Calls into a Conference

Format in C

unsigned int ***ctcConferenceJoin*** *(ctcChanId channel,*
unsigned int heldCallRefId,
unsigned int activeCallRefId,
*unsigned int *newCallRefId)*

Description

The ctcConferenceJoin routine merges two or more calls into a single conference call.

For example, for A to include B and C in a conference call, A has to:

1. Call B, using ctcMakeCall. B answers the call.
2. Call C, using ctcConsultationCall, which automatically puts the call to B on hold.
3. Invoke ctcConferenceJoin when connected and talking to C. A, B, and C are then in a conference call.

To include other parties in the conference, A simply repeats the sequence of ctcConsultationCall and ctcConferenceJoin for each additional party.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

heldCallRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the call reference identifier for the held call to be included in the conference. Specify the call reference identifier returned by ctcGetEvent or ctcWinGetEvent for the held call.

activeCallRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the call reference identifier for the active call to be included in the conference.

The call reference identifier for the active call is returned by ctcGetEvent or ctcWinGetEvent.

newCallRefId

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This argument is the address of a 32-bit integer into which is written a call reference identifier for the new conference call.

ctcConsultationCall
Make a Consultation Call

Format in C

```
unsigned int ctcConsultationCall (ctcChanId        channel,
                                  ctcDeviceString  calledNumber,
                                  unsigned int     consultType,
                                  unsigned int     callRefId,
                                  ctcApplString    applicationData,
                                  unsigned int     *newCallRefId)
```

Description

The ctcConsultationCall routine makes a call to a third party when there is a current call on the assigned device. You can then use one of the following routines:

- ctcTransferCall to transfer the call and disconnect the assigned device
- ctcConferenceJoin to join the original call and the call to the third party into a conference call

When you initiate a call transfer or conference call, always use ctcConsultationCall. This routine makes the telephone call and allows the switch to allocate any required resources.

Making a Conference Call

For a conference call, you use ctcConsultationCall only for the second call, or subsequent calls; you make the initial call using ctcMakeCall.

For example, for A to include B and C in a conference call:

1. A calls B, using ctcMakeCall.
2. A calls C, using ctcConsultationCall, which automatically puts the call to B on consultation hold.
3. A invokes ctcConferenceJoin when connected and talking to C. A, B, and C are then in a conference call.

To include other parties in the conference, A repeats the sequence of ctcConsultationCall and ctcConferenceJoin for each additional party.

Transferring a Call

To transfer a call, use ctcConsultationCall followed by ctcTransferCall.

For example, for A to transfer to C an incoming call from B (where A's current call is the call from B):

1. B calls A, using ctcMakeCall, and A answers.
2. A calls C, using ctcConsultationCall, which automatically puts the call from B on hold.
3. A invokes ctcTransferCall when connected to C. B and C are now connected, and A is automatically disconnected.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

calledNumber

type: **ctcDeviceString**
access: **read only**
mechanism: **by value**

This character string contains the number of the device you have called. The ASCII string can contain any combination of numbers 0 through 9 and the characters * and #.

The maximum length for calledNumber is specified by the literal ctcMaxDnLen in the CTC definitions file. Note that this maximum length includes the null termination character (NUL).

consultType

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This argument specifies the type of consultation call made to the third party. It contains one of the values in Table 2–1.

Table 2–1 Consult Type Values for ctcConsultationCall

Value	Description
ctcK_ConsultGeneric	The call is initiating either a call transfer or a conference call.
ctcK_ConsultTransfer	The call is initiating a call transfer.
ctcK_ConsultConference	The call is initiating a conference call.

The value that you specify depends on the switch you are using. For most switches, you can use ctcK_ConsultGeneric. However, some switches require you to specify whether the call is initiating a transfer or a conference call. If a switch does not accept ctcK_ConsultGeneric, it is noted in the switch-specific appendix.

callRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the call reference identifier for the current active call. Specify the call reference identifier for the active call as returned by ctcGetEvent, ctcWinGetEvent, or other associated routine such as, ctcMakeCall.

applicationData

type: **ctcApplString**
access: **read only**
mechanism: **by value**

This argument is the address of a NUL-terminated character string that you want to associate with a call. For example, customer reference information or account data.

If the consultation call is successful, the data is stored by the switch and reported on subsequent events until the call is terminated.

If you do not want to associate data with the call, pass a zero-length string.

newCallRefId

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This argument is the address of a 32-bit integer that receives the call reference identifier for the new call.

Note that some switches may not supply a call reference identifier with this argument. Refer to the switch-specific appendixes for details.

ctcDeassign
Deassign a Channel

Format in C

*unsigned int **ctcDeassign** (ctcChanId channel)*

Description

The ctcDeassign routine deassigns the channel from the device and frees all resources associated with it, both locally and on the CTC server.

Use this routine at the end of a user session; that is, when the user has finished using a CTC application and the application no longer needs to make use of the device to which the channel was assigned. Monitoring and routing are switched off before ctcDeassign completes. If you call ctcDeassign and there are outstanding monitoring or routing requests, a condition value is returned:

- For ctcGetEvent or ctcWinGetEvent, the ctcMonitorOff condition value is returned.
- For ctcGetRouteQuery or ctcWinGetRouteQuery, the ctcRoutingOff condition value is returned.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

ctcDeflectCall
Deflect a Ringing Call to Another Extension

Format in C

```
unsigned int ctcDeflectCall (ctcChanId          channel,
                             unsigned int       callRefId,
                             ctcDeviceString    destinationDn,
                             ctcApplString      applicationData)
```

Description

The ctcDeflectCall routine allows you to deflect a call ringing on the assigned device to another extension.

You use the destinationDn argument to specify the dialable number of the extension to which you want the call deflected. If the switch administrator has defined a default extension to which calls are deflected, the routine does not require the destinationDn argument.

Arguments

channel

type:	**ctcChanId**
access:	**read only**
mechanism:	**by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

callRefId

type:	**integer (unsigned)**
access:	**read only**
mechanism:	**by value**

This argument is a 32-bit integer that contains the call reference identifier for the ringing call. The call reference identifier for the ringing call is returned by ctcGetEvent or ctcWinGetEvent.

ctcDeflectCall

destinationDn

type: **ctcDeviceString**
access: **read only**
mechanism: **by value**

This character string contains the number of the destination device. The ASCII string can contain any combination of numbers 0 through 9 and the characters * and #.

The maximum length for destinationDn is specified by the literal ctcMaxDnLen in the CTC definitions file. Note that this maximum length includes the null termination character (NUL).

applicationData

type: **ctcApplString**
access: **read only**
mechanism: **by value**

You use this argument to associate data, for example, customer reference information or account details, with the call. The argument is the address of a NUL-terminated character string.

If the call is successful, the data is stored by the switch and reported on subsequent events until the call is terminated.

If you do not want to associate data with the call, pass a zero-length string.

ctcErrMsg
Get the Defined Name for a Condition Value

Format in C

char ***ctcErrMsg** (unsigned int errorCode)*

Description

The ctcErrMsg routine returns the address of a character string that contains the defined name associated with a condition value.

The defined name can provide you with more information by indicating the nature of the condition. For example, ctcMonitorOff indicates that monitoring is set off for a channel. You can also use the name to refer to Chapter 3 which describes CTC conditions.

Each condition value is associated with a name in the language-specific error definitions file (for example, CTC_ERR.H). When you use ctcErrMsg, CTC returns the address of a null-terminated character string that contains the defined name for a condition value.

For example, if you specify the value 1014 with the errorCode argument, CTC returns the address of a null-terminated character string that contains the name ctcInvLogId. You can then refer to Chapter 3 for a description of ctcInvLogId.

Arguments

errorCode

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the condition value returned by a CTC routine. CTC returns the address of a null-terminated character string that contains the name associated with this condition value.

If the routine cannot map a name to the condition value, it returns the address of a character string containing the decimal value of the input.

If you specify a condition value for an RPC error, the character string contains both:

- The name CTC associates with the condition
- The name RPC associates with the condition

ctcErrMsg

For example, ctcErrMsg can return the address of the character string ctcRpcConnecFail/rpc_s_ss_in_null_context. In this example, ctcRpcConnecFail is the CTC-defined name and rpc_s_ss_in_null_context is the RPC name associated with the condition.

ctcGetAgentStatus
Get Agent Status Information

Format in C

*unsigned int **ctcGetAgentStatus***	*(ctcChanId*	*channel,*
	unsigned int	**agentMode,*
	ctcDeviceString	*agentData)*

Description

The ctcGetAgentStatus routine returns information about the operating mode for an ACD agent.

The operating mode is set either manually or with the ctcSetAgentStatus routine.

Using this routine, you can find out if the Agent is:

- Logged into a queue
- Ready to take calls
- Busy
- Completing details after a call
- Doing other work

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

agentMode

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This argument is the address of a 32-bit integer into which CTC writes one of the agent mode values shown in Table 2–2.

Table 2–2 Agent Mode Values for ctcGetAgentStatus

Value	Description
ctcK_AgentReady	The agent is ready to receive calls
ctcK_AgentNotReady	The agent is not ready to receive calls
ctcK_AgentOtherWork	The agent cannot take calls because of other work
ctcK_AgentAfterCallWork	The agent is completing details of a call
ctcK_AgentLogout	The agent is logged out

agentData

type: **ctcDeviceString**
access: **write only**
mechanism: **by reference**

This argument receives optional data (such as a password). You must supply enough memory for CTC to return this data. The maximum length for agentData is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

ctcGetCallForward
Get Information About Call Forward

Format in C

unsigned int ***ctcGetCallForward***	*(ctcChanId*	*channel,*
	unsigned int	**forwardMode,*
	ctcDeviceString	*forwardDN)*

Description

This routine returns information about the current call forward setting for incoming calls on the assigned device. The call forward mode is set either manually or with the ctcSetCallForward routine.

ctcGetCallForward shows whether the following calls to the assigned device are redirected:

- External calls only
- Internal calls only
- All calls
- No calls

This routine also shows whether incoming calls are forwarded if the assigned device is busy or if the call is not answered after a period of time (as determined by the switch).

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

ctcGetCallForward

forwardMode

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This argument is a 32-bit integer into which CTC writes one of the values shown in Table 2–3.

Table 2–3 ctcGetCallForward Modes Returned

Value	Calls Forwarded
ctcK_CfAll	All calls
ctcK_CfExtBusy	External calls when the assigned device is busy
ctcK_CfExtNoAnswer	External calls when there is no answer from the assigned device
ctcK_CfIntBusy	Internal calls when the assigned device is busy
ctcK_CfIntNoAnswer	Internal calls when there is no answer
ctcK_CfNoAnswerBusy	All calls if there is no answer or the assigned device is busy

forwardDN

type: **ctcDeviceString**
access: **write only**
mechanism: **by reference**

This argument is the address of a character string used by CTC to return the number of the destination device. If call forwarding is not set, CTC returns a zero-length character string.

The maximum length for forwardDN is specified by the literal ctcMaxDnLen in the CTC definitions file. Note that this maximum length includes the null termination character (NUL).

ctcGetChannelInformation
Get Information About a Channel

Format in C

unsigned int ***ctcGetChannelInformation*** *(ctcChanId channel,*
*ctcChanData *channelData)*

Description

The ctcGetChannelInformation routine returns information about the communications channel and the device to which the channel is assigned. The routine can provide the following information:

- The line type (telephone, data set, trunk, group, Voice Response Unit (VRU), route point, or monitor channel)
- The prime number of the device (the extension number or trunk number for the device, or the prime number on a multiline set)
- The device type (if not ACD), for example, 500, 2500, or a feature phone
- The CTC procedures supported
- The DN for the device, for example, its telephone number or extension number

You need to use ctcGetChannelInformation only once, each time you assign a channel to a device; it provides static information on the nature of the device and the channel that is assigned to that device.

Monitor Channels

For monitor channels, ctcGetChannelInformation returns the following information only:

- The CTC procedures supported
- A device number for the monitor channel

Use the device number (returned in the setDN field of the ctcChanData structure) with ctcAddMonitor to set up monitoring a monitor channel. For more information, see the description of ctcAddMonitor.

Note that you must specify the device number exactly as it is provided using the same combination of numbers and **uppercase** letters. For example, 1E4DC0.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

channelData

type: **ctcChanData**
access: **write only**
mechanism: **by reference**

This argument contains the address of a fixed-format structure, for which you allocate memory of type ctcChanData. The structure is defined in a CTC definitions file (see Section 1.5) and is formatted as follows:

```
ctcChanData {
   unsigned int       lineType;
   unsigned int       prime;
   unsigned int       setType;
   unsigned int       procedureSupport;
   unsigned int       attributeSupport;
   ctcDeviceString    setDN;
   unsigned int       switchSpecificSupport;
};
```

The following information is returned in the ctcChanData fields:

- **lineType**

  This 32-bit integer contains a value that identifies the type of line. The following table shows the values that can be returned:

Value	Description
ctcK_LineACD	The channel is assigned to a group on the switch
ctcK_LineDataSet	The channel is assigned to a data set
ctcK_LineMonitorChannel	The channel is assigned to a monitor channel
ctcK_LineRoutePoint	The channel is assigned to a route point on the switch
ctcK_LineTrunk	The channel is assigned to a trunk line on the switch
ctcK_LineVoiceSet	The channel is assigned to a telephone

- **prime**

  This 32-bit integer contains a value that defines whether the line is a primary line (1) or not (0). The primary line is a line selected by the user to which a device is automatically connected when it goes off-hook (that is, when it enters the initiate state).

- **setType**

  This 32-bit integer contains a value that identifies the type of telephone set associated with the assigned channel. For details of the values returned in this field, refer to the switch-specific appendixes.

- **procedureSupport**

  This 32-bit integer identifies the procedure routines supported by the switch for the assigned device. The following values can be returned in this field:

  ctcM_AddMonitor
  ctcM_AnswerCall
  ctcM_Assign
  ctcM_AssociateData
  ctcM_CancelCall
  ctcM_ConferenceJoin
  ctcM_ConsultationCall
  ctcM_Deassign
  ctcM_DeflectCall
  ctcM_GetChannelInformation
  ctcM_GetEvent
  ctcM_GetRouteQuery
  ctcM_HangupCall
  ctcM_HoldCall
  ctcM_MakeCall
  ctcM_MakePredictiveCall
  ctcM_PickupCall
  ctcM_ReconnectHeld
  ctcM_RemoveMonitor
  ctcM_RespondToInactive
  ctcM_RespondToRouteQuery
  ctcM_RetrieveHeld
  ctcM_SendDTMF
  ctcM_SingleStepTransfer
  ctcM_Snapshot
  ctcM_SwapWithHeld
  ctcM_TransferCall

Note that:

- If ctcM_GetEvent is returned, the switch supports ctcGetEvent and ctcWinGetEvent.
- If ctcM_GetRouteQuery is returned, the switch supports ctcGetRouteQuery and ctcWinGetRouteQuery.
- A function-supported mask is not returned for ctcErrMsg. This routine is supported for all channels.

• **attributeSupport**

This 32-bit integer identifies the attribute routines supported by the switch for the assigned device. Attribute routines are routines that set operating modes for the assigned device.

The following values can be returned:

ctcM_GetAgentStatus
ctcM_GetCallForward
ctcM_GetDoNotDisturb
ctcM_GetMessageWaiting
ctcM_GetMonitor
ctcM_SetAgentStatus
ctcM_SetCallForward
ctcM_SetDoNotDisturb
ctcM_SetMessageWaiting
ctcM_SetMonitor

• **setDN**

This field contains one of the following:

- If the channel is assigned to a telephony device or route point, the number associated with the device or route point, for example, its telephone number or extension number.
- If the channel is assigned to a monitor channel, a device number generated by CTC. You use this device number with the ctcAddMonitor command to set up a monitor channel that monitors this monitor channel.

The maximum length for setDN is specified by the literal ctcMaxDnLen, in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

- **switchSpecificSupport**

  This 32-bit integer identifies any additional switch-specific CTC routines that are supported. These are routines provided as extensions to the CTC API.

  Refer to the switch-specific appendixes for more information about the values returned in this field.

  Note that no values are returned in this field if, when you assigned the channel, you specified the value ctcK_None in the APIextensions field of the ctcAssignData structure.

ctcGetDoNotDisturb
Get Information About Do Not Disturb

Format in C

unsigned int ***ctcGetDoNotDisturb*** *(ctcChanId channel,*
*unsigned int *DNDMode)*

Description

This routine returns information about the Do-Not-Disturb setting for the assigned device. It provides information on the current setting as set by the user, either manually or with the ctcSetDoNotDisturb routine.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

DNDMode

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This argument is the address of a 32-bit integer into which CTC writes one of the values in the following table:

Value	Description
ctcK_On	Indicates that Do-Not-Disturb on the assigned device is set on
ctcK_Off	Indicates that Do-Not-Disturb on the assigned device is set off

ctcGetEvent
Get Information About Event and State Changes

Format in C

unsigned int ***ctcGetEvent*** *(ctcChanId channel,*
*ctcEventData *eventData,*
unsigned int dontWait)

Description

The ctcGetEvent routine returns information on telephone activity on the assigned device, or on devices associated with a monitor channel.

It can return details of:

- Call states
- Call events
- Agent status events
- Call types
- Call reference
- Other parties involved in the telephone call
- Application data stored with the call
- Devices monitored on a monitor channel
- Time and date at which an event occurred

The amount of information that CTC returns depends on the information provided by the switch. This may be different for a call that is internal to the switch and for an outside call, depending on the type of trunks connected to the switch.

Calling ctcGetEvent

For all assigned devices **except monitor channels**, you must set monitoring on with the ctcSetMonitor routine before you use this routine.

Note that if you post a ctcGetEvent request and the previous ctcGetEvent request has not yet completed, a ctcEventInProgress error is returned.

ctcGetEvent

Restriction

ctcGetEvent returns information only when there is call activity. Dialogic recommends you use a multithreaded program to call this routine so that your application can continue (see Section 1.9.4).

This does not apply to Windows 3.1/3.11 applications. To return telephone activity and call other routines, Windows 3.1/3.11 applications must use ctcWinGetEvent. See the description of ctcWinGetEvent for more information.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

eventData

type: **ctcEventData**
access: **write only**
mechanism: **by reference**

This argument contains the address of the following fixed-format structure:

```
ctcEventData{
   unsigned int     refId;
   unsigned int     netCallId;
   unsigned int     oldRefId;
   unsigned int     oldNetCallId;
   unsigned int     state;
   unsigned int     event;
   unsigned int     eventQualifier;
   unsigned int     type;
   unsigned int     otherPartyType;
   unsigned int     otherPartyQualifier;
   ctcDeviceString  otherParty;
   unsigned int     otherPartyTrunk;
   unsigned int     otherPartyGroup;
   unsigned int     thirdPartyType;
   unsigned int     thirdPartyQualifier;
   ctcDeviceString  thirdParty;
   unsigned int     thirdPartyTrunk;
   unsigned int     thirdPartyGroup;
   unsigned int     calledPartyType;
   unsigned int     calledPartyQualifier;
   ctcDeviceString  calledParty;
   unsigned int     calledPartyTrunk;
   unsigned int     calledPartyGroup;
```

```
    ctcApplString    applicationData;
    ctcDeviceString  monitorParty;
    ctcDeviceString  nestedMonitorChannel;
    unsigned int     agentMode;
    ctcDeviceString  agentId;
    ctcDeviceString  agentGroup;
    ctcDeviceString  agentData;
    ctcDeviceString  logicalAgent;
    ctcDeviceString  dtmfDigits;
    unsigned int     originatingPartyType;
    unsigned int     originatingPartyQualifier;
    ctcDeviceString  originatingParty;
    unsigned int     originatingPartyTrunk;
    unsigned int     originatingPartyGroup;
    unsigned int     secOldRefId;
    unsigned int     callsQueued;
    ctcAccountInfo   accountInfo;
    ctcTimeStamp     timeStamp;
    unsigned int     privateData;
};
```

The strings in this structure are all null-terminated.

The ctcEventData structure contains the following:

- **refId**

  This field contains the call reference identifier for a particular call. Use this call reference when you use CTC routines that affect existing calls. For example, when you use CTC routines to transfer a call, cancel a call, or create a conference call.

  Note that the call reference is supplied by the switch but does not necessarily remain constant for the duration of the call. For example, if party A answers an incoming call and then transfers the call to party B, the call reference reported for A (the original call) and B (the transferred call) may not be the same. Do not rely on this mechanism to trace calls from one party to another.

- **netCallId**

  This 32-bit field returns a network call identifier. This is used to identify a call that is handled by more than one switch in a network of switches. For example, for overflow calls.

  Information in the netCallId field is returned only if the switch can supply the information.

- **oldRefId**

  If the reference identifier for a call changes, this field contains the previous call reference. A call reference can change if there is new telephony activity on the assigned device. For example, if the call on the assigned device is included in a conference call.

  Information in the oldRefId field is returned only if the switch can supply the information.

- **oldNetCallId**

  This 32-bit field returns the previous network call identifier. This is used to identify which switch first receives a call when the call is handled by more than one switch.

  Information in the oldNetCallId field is returned only if the switch can supply the information.

- **state**

  This 32-bit field contains a value that identifies the state of the current call. Table 2–4 shows the possible states of a call, and the corresponding values returned. These call state literals are supplied in a CTC definitions file (see Section 1.5).

  Example state transitions for an outbound call are as follows:

  Null/Initiate → **Deliver/Fail** → **Active** → **Initiate/Null**

  For a typical incoming call, the state transitions are as follows:

  Null → **Receive** → **Active** → **Initiate/Null**

  Note that if a call you are monitoring is routed over a trunk line, you do not necessarily see the Active state on an outbound call (depending on the type of trunk connected to the switch).

  Refer to the switch-specific appendixes for more information about the call states supplied by particular switches, and for details of the states returned for different events.

Table 2–4 Call States Returned by ctcGetEvent

State	Value	Description
Active	ctcK_ActiveState	The call is active.
Deliver	ctcK_DeliverState	The user has finished dialing and is waiting for an answer from the destination device.
Fail	ctcK_FailState	The switch could not complete the call because, for example, the user has dialed a busy device or a nonexistent number, or there are insufficient switch resources available to complete the call.
Hold	ctcK_HoldState	The call has been put on hold.
Initiate	ctcK_InitiateState	An outbound call is being placed. Typically, the assigned device is off-hook and receiving a dial tone.
Null	ctcK_NullState	Signals the end of a call. The Null state is also known as Idle.
Queued	ctcK_QueueState	A call has entered the group queue you are monitoring, or an outbound call from the assigned device has been queued.
Receive	ctcK_ReceiveState	The assigned device has received a call, and is ringing.
Unavailable	ctcK_UnavailableState	The assigned device is unavailable, because, for example, the user has left the telephone off-hook for too long, or the telephone is in maintenance.

- **event**

  This 32-bit integer identifies the call event. Table 2–5 shows the possible agent event values returned, and Table 2–6 shows the possible call event values returned.

  When ctcGetEvent returns the information, compare the values returned in the integer with the call event literals supplied in a CTC definitions file (see Section 1.5).

Table 2–5 Agent Events Returned by ctcGetEvent

Event-Value	Description
ctcK_AgentLoggedOn	The agent has logged in.
ctcK_AgentLoggedOff	The agent has logged out.
ctcK_AgentModeChange	The work mode for an agent has changed. Check the agentMode field for a value that identifies the new work mode.

Table 2–6 Call Events Returned by ctcGetEvent

Event-Value	Description
ctcK_BackInService	The device has returned to service.
ctcK_CallInformation	The account information or authorization information associated with a call has changed.
ctcK_DestBusy	The destination device is busy (engaged).
ctcK_DestChanged	The call from the assigned device was not answered on the original destination and has been redirected to another destination. For example, if the original destination had set call forward, the call would not have been presented at the original destination but would have been routed directly to the new destination.
ctcK_DestInvalid	The attempted call has failed because the destination device is incompatible. For example, making a call from a voice set to a data set.
ctcK_DestNotObtainable	The call could not be completed, probably because the wrong number was dialed.
ctcK_DestSeized	A call has been successfully dialed. If this call is external to the ACD, the network number has been verified and the outbound trunk seized. This does not indicate that the other end is actually ringing or answered.
ctcK_Diverted	An incoming call on the assigned device has been diverted to another destination.
ctcK_Error	The call has failed for an unspecified reason. For more information, check the eventQualifier field. (Refer to the switch-specific appendixes for a description of event qualifiers.)

Table 2–6 Call Events Returned by ctcGetEvent (Continued)

Event-Value	Description
ctcK_InboundCall	A new call has arrived at the assigned device.
ctcK_Offhook	A new call has been made from the assigned device.
ctcK_OffhookPrompt	This event indicates one of the following: • You have tried to place a call from a 2500 set but you need to take the phone off-hook for the call to continue • The user invoked ringback, and the switch is signaling that the callback has matured • The switch is signaling because you kept a call on hold for too long, or have hung up with a call on hold
ctcK_OpAnswered	The other party answered the call from the assigned device.
ctcK_OpConferenced	Another party on the call has created a conference call.
ctcK_OpDisconnected	The other party hung up the call.
ctcK_OpHeld	The other party has placed the call on hold.
ctcK_OpRetrieved	The other party has retrieved the held call.
ctcK_Other	An event has occurred. This event is not significant for basic call processing and can be generated by a number of possible causes. See the event qualifier item for further information on this event, but remember the event qualifiers are switch-dependent.
ctcK_OutOfService	The device is out of service.
ctcK_Private	Data has been sent by the switch. The type of data sent is specific to the switch manufacturer.
ctcK_TpAnswered	This party has answered an incoming call on the assigned device.
ctcK_TpConferenced	This party has included another party in a conference call.
ctcK_TpDisconnected	This party has disconnected the current call.
ctcK_TpRetrieved	This party has retrieved a call that was either on hold or in the call-waiting queue.
ctcK_TpSuspended	This party has placed a call on hold.

Table 2–6 Call Events Returned by ctcGetEvent (Continued)

Event-Value	Description
ctcK_Transferred	The call has been transferred from another device. This event is returned to both parties on the new call.
ctcK_Unavailable	The device has become unavailable (out of service). This could be: • A temporary condition, for example, because the device has remained off-hook and listening to a dial tone for too long • A more permanent condition if the device is in maintenance

- **eventQualifier**

  This 32-bit integer provides more detailed information on certain events. For a description of each event qualifier, see the switch-specific appendixes.

  When ctcGetEvent returns the information, compare the values returned in the integer with the event qualifier literals supplied in a CTC definitions file (see Section 1.5).

- **type**

  This 32-bit integer identifies the type of call in progress. The call type helps to clarify the result of the call event and state.

  For a description of each call type, see the switch-specific appendixes.

  When ctcGetEvent returns the information, compare the values returned in the integer with the event qualifier literals supplied in a CTC definitions file (see Section 1.5).

- **Other Party**

The other party fields return information about the party that the user of the assigned device is calling or to which they are connected. Table 2–7 describes the information returned in the other party fields.

Table 2–7 Other Party Information

Field	Description
otherPartyType	This 32-bit field identifies the number for the other party as a Calling Line ID (CLID, DN, or Dialed Number Identification Service (DNIS). It contains one of the following values: ctcK_LineId ctcK_Dn ctcK_Dnis
otherParty	This field contains the CLID, DN, or DNIS for the other party. The maximum length for the CLID, DN, or DNIS, is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).
otherPartyQualifier	This 32-bit field provides additional information about the other party. For more information, refer to the switch-specific appendixes.
otherPartyTrunk	This 32-bit field contains the trunk line number from the switch.
otherPartyGroup	This 32-bit field contains the trunk group number.

- **Third Party**

  The third party fields return information about any third party involved with the current call on the assigned device. Table 2–8 describes the third party fields.

Table 2–8 Third Party Information

Field	Description
thirdPartyType	This 32-bit field identifies the number for the third party as a CLID, DN, or DNIS. It contains one of the following values: ctcK_LineId ctcK_Dn ctcK_Dnis
thirdParty	This field contains the CLID, DN, or DNIS for the third party. The maximum length for the CLID, DN, or DNIS is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).
thirdPartyQualifier	This 32-bit field provides additional information about the third party. For details of the information supplied in this field, see your switch-specific appendix.
thirdPartyTrunk	This 32-bit field contains the trunk line number from the switch.
thirdPartyGroup	This 32-bit field contains the trunk group number.

- **Called Party**

  The called party fields return information about the party originally called. For example, if A calls B, and then B transfers the call to C (the monitored device), the called party is B.

  Table 2–9 describes the called party fields.

Table 2–9 Called Part Information

Field	Description
calledPartyType	This 32-bit field identifies the number for the called party as a CLID, DN, or DNIS. It contains one of the following values: ctcK_LineId ctcK_Dn ctcK_Dnis
calledParty	This field contains the CLID, DN, or DNIS for the called party. The maximum length for the CLID, DN, or DNIS is specified by the literal ctcMaxDnLen, in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).
calledPartyQualifier	This 32-bit field provides additional information about the called party. For more information, see your switch-specific appendix.
calledPartyTrunk	This 32-bit field contains the trunk line number from the switch.
calledPartyGroup	This 32-bit field contains the trunk group number.

  CTC sometimes returns a null datatype for the other party, third party, and called party fields. In such cases, the switch cannot identify the parties involved in the call. Check the switch-specific appendixes for more information.

- **applicationData**

  This field returns data that has been associated with a call (for example, by the ctcMakeCall routine) and stored by the switch. The data is returned as a character string.

  Refer to the switch-specific appendixes to check whether your switch supports application data.

  The maximum length for applicationData is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

- **monitorParty**

  Information is returned in this field for monitor channels. You can assign to a monitor channel to receive events for a number of devices on a single channel (see ctcAssign for more information).

  The device number returned in this field identifies the device for which event information is returned.

  The maximum length for monitorParty is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

- **nestedMonitorChannel**

  This field returns a device number that identifies the nested monitor-channel for which event information is returned. A nested monitor-channel is a channel that is monitored by another monitor channel.

  The maximum length for nestedMonitorChannel is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

- **agentMode**

  This 32-bit field returns the current work mode for an agent. It can contain one of the following values:

  ctcK_AgentReady
  ctcK_AgentNotReady
  ctcK_AgentOtherWork
  ctcK_AgentAfterCallWork
  ctcK_AgentLogin
  ctcK_AgentLogout

  Check the switch-specific appendixes for details of any other, switch-specific values that can be returned in this field.

- **agentId**

  When the channel is assigned to a queue, this field returns the identifier (ID) (for example, an extension number) for an agent associated with that queue. Refer to the switch-specific appendixes to check whether your switch supports agent IDs.

  The maximum length for agentId is specified by the literal ctcMaxDnLen in a

CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

- **agentGroup**

  This field contains the DN for a group (ACD group or queue). Refer to the switch-specific appendixes to check whether your switch returns information in this field.

  The maximum length for agentGroup is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

- **agentData**

  This field returns agent data, for example, an agent's password. Refer to the switch-specific appendixes to check whether your switch returns information in this field.

  The maximum length for agentData is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

- **logicalAgent**

  This field returns the DN for a logical agent. Refer to the switch-specific appendixes to check whether your switch returns information in this field.

  The maximum length for logicalAgent is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

- **dtmfDigits**

  For channels assigned to route points, this field returns DTMF digits that are collected as a call is routed. Refer to the switch-specific appendixes to check whether your switch returns DTMF digits in this field.

  The maximum length for dtmfDigits is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

- **Originating Party**

  The originating party fields contain details of:

  - The point of entry on the switch for an inbound call
  - The point of exit on the switch for an outbound call

For example, the trunk number on which the switch received an inbound call.

CTC returns information in the originating party fields only if:

– The state field contains the value ctcK_ReceiveState or ctcK_DeliverState

– The switch can provide the information

For details of support, check the switch-specific appendixes.

Table 2–10 describes the originating party fields.

Table 2–10 Originating Party Fields

Field	Description
originatingPartyType	This 32-bit field identifies the originating number as a CLID, DN, or DNIS. It contains one of the following values: ctcK_LineId ctcK_Dn ctcK_Dnis
originatingPartyQualifier	This 32-bit field provides additional information about the originating party. For more information, refer to the switch-specific appendixes.
originatingParty	This field contains the actual CLID, DN, or DNIS for the originating party. The maximum length for the CLID, DN, or DNIS, is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).
originatingPartyTrunk	If the originating party is a trunk, this 32-bit field contains the trunk line number as defined on the switch.
originatingPartyGroup	If the originating party is a trunk group, this 32-bit field contains the trunk group number as defined on the switch.

- **secOldRefId**

  If the reference identifier for a call changes because it has been transferred or merged in a conference call, this field contains the call reference for the consultation call.

  For example, if A calls B then includes C in a conference call, the following call reference identifiers are returned when the conference call has been established:

This field...	Contains...
refId	The call reference for the conference call between A, B, and C
oldRefId	The call reference for the original call made by A to B
secOldRefId	The call reference for the consultation call made by A to C

- **callsQueued**

  If the call is placed in a queue, this 32-bit field can return the total number of calls in the queue.

  This value is returned only if it can be provided by the switch. Check the switch-specific appendixes for details.

- **accountInfo**

  This field contains 32 bytes of free-format data. Refer to your switch manufacturer for details of the type of account information that can be returned.

- **timeStamp**

  By default, this field contains the date and time the CTC server received the event. It can also return the date and time that the switch processed the event, if the communications link is configured to return this information (for details, refer to the *CT-Connect Installation and Administration Guide* for your CTC server platform) and if your switch can provide the information (refer to the switch-specific appendixes for details).

  It contains a fixed-format structure:

```
ctcTimeStamp{
      short        year;
      short        month;
      short        day;
      short        hour;
      short        minute;
      short        second;
      short        millisec;
      short        mindiff;
      unsigned     intutc;
    };
```

The following table describes the fields in the ctcTimeStamp structure:

Field	Description
year	Four-digit value identifying the year. For example, 1998.
month	A value from 1 through 12.
day	A value from 1 through 31.
hour	A value from 0 through 23.
minute	A value from 0 through 59.
second	A value from 0 through 59.
millisec	A value from 0 through 999.
mindiff	Minimum differential between the current time and GMT. A value is returned in this field for UTC time only.
utc	A non-zero value in this field indicates that the structure provides UTC time. If this field is empty, the structure provides absolute time.

- **privateData**

  A value in this field indicates that CTC has received from the switch data relating to the event. The type of data returned is specific to individual switches.

  For this version of CTC, private data is supported by CSTA switches only. To retrieve the data, you use the ctcCstaGetPrivateEventData routine before reposting ctcGetEvent. Refer to Appendix B for details.

  For other switches, the privateData field returns null data.

dontWait

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer is a Boolean value which, when set, allows an application to poll for events without having to create a separate thread. If there is no new event data, the routine will not block and a ctcNoEvent condition value is returned.

States and Events for Groups

If you have assigned a channel to a group queue on the switch, CTC can provide information on:

- The call reference for each call joining the queue
- Calls leaving the queue
- The identity of the other party (DN, Calling Line ID, or trunk) when available, for calls entering or leaving the queue
- The destination for a call leaving the queue

Table 2–11 lists the states and events that CTC returns if you are monitoring a channel assigned to a group (queue).

Table 2–11 Queue Monitoring

State	Event	Cause
Null	ctcK_OpDisconnected	The caller has hung up.
Queued	ctcK_InboundCall	A call has joined the queue.
Queued	ctcK_Diverted	The call has been moved to another queue.
Deliver	ctcK_DestSeized	The call has been delivered to an ACD agent.
Active	ctcK_OpAnswered	The agent has answered.

Error for Event Data Lost

If a number of events occur at the same time, it is possible for the CTC server to lose an event message. Although this is unlikely, the CTC server will return a ctcEventDataLost error for any messages lost.

ctcGetMessageWaiting
Get Information About the Message Waiting Indicator

Format in C

*unsigned int **ctcGetMessageWaiting** (ctcChanId channel,*
*unsigned int *messageWaitingMode)*

Description

This routine returns information about the current setting for the message waiting indicator. This indicator is usually a lamp on the telephone set which is lit if there is a message waiting.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

messageWaitingMode

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This argument is the address of a 32-bit integer that receives one of the values in the following table:

Value	Description
ctcK_On	Indicates that the message waiting indicator for the assigned device is set on
ctcK_Off	Indicates that the message waiting indicator for the assigned device is set off

ctcGetMonitor
Get Information About the Monitoring State

Format in C

unsigned int ***ctcGetMonitor*** *(ctcChanId channel,*
*unsigned int *monitorMode)*

Description

This routine returns information about the current monitoring state of the assigned device. The monitoring state can be changed with the ctcSetMonitor routine.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

monitorMode

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This argument is the address of a 32-bit integer that receives one of the values in the following table:

Value	Description
ctcK_On	Indicates that monitoring on the assigned device is set on
ctcK_Off	Indicates that monitoring on the assigned device is set off

ctcGetRouteQuery
Get Route Query Messages from the Switch

Format in C

```
unsigned int ctcGetRouteQuery (ctcChanId       channel,
                               ctcRouteData    *routeData,
                               unsigned int    dontWait)
```

Description

The ctcGetRouteQuery routine requests that the switch passes a route request to your application whenever a call reaches the assigned route point.

When a call comes in at the assigned route point, ctcGetRouteQuery presents the call to your application so that the call can be routed. If supplied by the switch, this routine also returns information on the other parties involved in the call and the number dialed by the caller:

- For calls internal to the switch, this routine returns the extension number for the calling party.
- For an external call (connected to a central office or to another networked switch through a trunk), this routine returns the following information (if the switch and telephone network can supply the information):
  - The group number (for the trunk in use on the switch)
  - The calling line ID, which identifies the device that originated the call
  - The dialed number
  - Application data passed with the call

Call Routing

Call routing allows the application to redirect incoming calls for a route point to another destination. The application assigns a channel to a route point (the route point has a dialable number associated with it, but there is no real, physical device). Whenever a call reaches this route point, the switch passes a route request to the CTC application.

When the switch passes a route request, ctcGetRouteQuery returns data relating to the call, including a route reference. The application can then decide on a new destination for the call and supply this and the route reference as parameters to ctcRespondToRouteQuery. If the new destination is valid, the call is redirected.

Choosing Not to Reroute a Call

If you do not want the application to reroute a call presented by ctcGetRouteQuery, post a call to the ctcRespondToRouteQuery routine and specify the address of a zero-length character string with the newCalledNumber argument.

How to Call ctcGetRouteQuery

Use the following sequence of routines:

1. ctcAssign to assign the channel to a route point acting as a device. The route point has a DN associated with it, but there is no real, physical device.
2. If your switch allows you to enable or disable call routing for a route point, ctcSetRoutingEnable to enable routing.
3. ctcGetRouteQuery to be notified of route requests for calls to the assigned route point.
4. ctcRespondToRouteQuery to supply a route for the call.

For more information, refer to the descriptions of the ctcSetRoutingEnable and ctcRespondToRouteQuery routines.

ctcRouteInProgress

If you post a ctcGetRouteQuery request before the previous ctcGetRouteQuery request has completed, a ctcRouteInProgress error is returned.

Restriction

ctcGetRouteQuery returns only when it receives a route request from the switch. Dialogic recommends you use a multithreaded program to call this routine so that your application can continue to call other routines (see Section 1.9.4).

This does not apply to Windows applications. To return route query data and continue calling other routines, Windows applications must use ctcWinGetRouteQuery. See the description of ctcWinGetRouteQuery for more information.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

routeData

type: **ctcRouteData**
access: **write only**
mechanism: **by reference**

This argument contains the address of a fixed-format structure, for which you allocate memory of type ctcRouteData. The structure is defined in a CTC definitions file (see Section 1.5) and is formatted as follows:

```
ctcRouteData{
    unsigned int     routeId;
    unsigned int     refId;
    unsigned int     spare001;
    unsigned int     otherPartyType;
    ctcDeviceString  otherParty;
    unsigned int     otherPartyTrunk;
    unsigned int     otherPartyGroup;
    unsigned int     thirdPartyType;
    ctcDeviceString  thirdParty;
    unsigned int     thirdPartyTrunk;
    unsigned int     thirdPartyGroup;
    unsigned int     calledPartyType;
    ctcDeviceString  calledParty;
    unsigned int     calledPartyTrunk;
    unsigned int     calledPartyGroup;
    ctcApplString    applicationData;
    ctcDeviceString  dtmfDigits;
    ctcTimeStamp     timeStamp;
    unsigned int     privateData;
 };
```

CTC returns information in one or more of the fields described in Table 2–12.

Table 2–12 Information Returned by ctcGetRouteQuery

Field	Contents
routeId	The route identifier for the call to be routed. Use this identifier to specify a new route for the call with ctcRespondToRouteQuery.
refId	The call reference returned by ctcGetEvent.
spare001	Null data.
otherPartyType	A value that identifies the number of the other party as a CLID, DN, or DNIS.
otherParty	The CLID, DN, or DNIS for the other party.
otherPartyTrunk	A trunk line number from the switch for the other party.
otherPartyGroup	A trunk group number from the switch for the other party.
thirdPartyType	A value that identifies the number of an additional party on the call as a CLID, DN, or DNIS.

Table 2–12 Information Returned by ctcGetRouteQuery (Continued)

Field	Contents
thirdParty	The CLID, DN, or DNIS of an additional party involved in the call.
thirdPartyTrunk	A trunk line number from the switch for the third party.
thirdPartyGroup	A trunk group number from the switch for the third party.
calledPartyType	A value that identifies the number originally dialed as a CLID, DN, or DNIS.
calledParty	The CLID, DN, or DNIS originally dialed.
calledPartyTrunk	A trunk line number from the switch for the called party.
calledPartyGroup	A trunk group number from the switch for the called party.
applicationData	Data associated with the call to be routed.
dtmfDigits	DTMF digits collected as the call is routed.
timeStamp	Time the route request was processed either by the CTC server or by the switch (depending on the configuration of the link).
privateData	A value to indicate whether CTC has received private data from the switch.

For detailed information about the values returned in these fields, refer to the description of ctcGetEvent. For details of the fields supported by your switch, refer to the switch-specific appendixes.

dontWait

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer is a Boolean value which, when set, allows an application to poll for information without having to create a separate thread. If there is no new route data, the routine will not block and a ctcNoRoute status value is returned.

ctcGetRoutingEnable
Get the Routing State for a Device

Format in C

unsigned int ***ctcGetRoutingEnable*** *(ctcChanId channel,*
*unsigned int *routingMode)*

Description

The ctcGetRoutingEnable routine returns the routing state for the assigned route point:

- If routing is enabled, the switch passes route requests to CTC when it receives calls for the route point.
- If routing is disabled, the switch does not pass route requests to CTC when it receives calls for the route point.

To enable or disable routing for a route point, use ctcSetRoutingEnable. Refer to the description of ctcSetRoutingEnable for more information.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype and identifies the route point for which you want to receive route requests.

Specify the channel identifier returned by ctcAssign for the route point.

routingMode

type: **integer (unsigned)**
access: **write only**
mechanism: **by value**

This argument is the address of a 32-bit integer that receives one of the values in the following table:

This value...	Indicates that routing is...
ctcK_On	Enabled
ctcK_Off	Disabled

ctcHangupCall
Disconnect a Call

Format in C

*unsigned int **ctcHangupCall** (ctcChanId channel,*
unsigned int callRefId)

Description

The ctcHangupCall routine clears the active call on the assigned device, and returns the device to the null state.

Note that if you make a consultation call and then use ctcHangupCall, the original call may be transferred, or the switch may recall you. To end a consultation call and reconnect to the original party, use ctcReconnectHeld.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

callRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

The callRefId argument contains an optional value. If you have set monitoring on and can therefore identify the call reference, specify the call reference identifier for the assigned device returned by ctcGetEvent for ctcWinGetEvent, or the associated routine such as ctcMakeCall.

If you have not set monitoring on, you cannot specify a valid call reference value for the call you wish to hang up. Specify zero and the switch will hang up the current active call on the assigned device.

ctcHoldCall
Put Current Call on Hold

Format in C

*unsigned int **ctcHoldCall** (ctcChanId channel,*
unsigned int callRefId)

Description

The ctcHoldCall routine puts the current call on the assigned device on consultation hold. Calls placed on consultation hold can be included in a conference call or transferred to another extension. If the call is in the initiate state, use ctcRetrieveHeld to retrieve the held call. If the user has made another call after placing the original call on hold, use ctcReconnectHeld to end the consultation call and reconnect to the held call.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

callRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the call reference identifier for the call you are placing on hold. The call reference identifier is a value returned by ctcGetEvent or ctcWinGetEvent for the active call (the call to be held).

ctcMakeCall
Make a Call

Format in C

unsigned int ***ctcMakeCall*** *(ctcChanId*		*channel,*
	ctcDeviceString	*calledNumber,*
	ctcApplString	*applicationData,*
	unsigned int	**callRefId)*

Description

The ctcMakeCall routine makes a call from the device to which the channel is assigned to any number that the switch recognizes as valid.

You identify the device that you want to call with the calledNumber argument. This argument specifies the dialable number of the device.

On-hook Dialing

Some switches provide on-hook dialing. This means that a user can initiate a call and place a call without lifting the handset. The associated telephone must be a feature phone with a loudspeaker, and the switch must be able to set the telephone off-hook. The steps for on-hook dialing are as follows:

1. The user types the destination number at the keyboard, without lifting the handset.
2. The switch makes the connection to the destination telephone, and rings the originating telephone.
3. The switch sets the telephone off-hook, and the destination telephone starts ringing.

Limited On-Hook Dialing

Some switches provide a limited form of on-hook dialing for users with standard (nonfeature) telephones. The steps for limited on-hook dialing are as follows:

1. The caller dials from the keyboard without picking up the handset.
2. The switch makes the connection to the destination telephone, and rings the originating telephone.
3. The caller picks up the handset, and the destination telephone starts ringing.
4. If the switch cannot signal the telephone, the user must take the device off-hook before making the call.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

calledNumber

type: **ctcDeviceString**
access: **read only**
mechanism: **by reference**

This argument is the address of a character string that contains the number of the device you want to call. The ASCII string can contain any combination of numbers 0 through 9 and the characters * and #.

The maximum length for calledNumber is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

applicationData

type: **ctcApplString**
access: **read only**
mechanism: **by value**

You use this argument to associate data, for example, customer reference information or account details, with the call. The argument is the address of a NUL-terminated character string.

If the call is successful, the data is stored by the switch and reported on subsequent events until the call is terminated.

If you do not want to associate data with the call, pass a zero-length string.

callRefId

type: **integer (unsigned)**
access: **write only**
mechanism: **by value**

This argument is the address of a 32-bit integer that receives the call reference identifier for the call.

ctcMakePredictiveCall
Make Predictive Calls

Format in C

```
unsigned int ctcMakePredictiveCall (ctcChanId        channel,
                                    ctcDeviceString  calledNumber,
                                    unsigned int     allocation,
                                    ctcApplString    applicationData,
                                    unsigned int     *callRefId,
                                    unsigned int     numberOfRings)
```

Description

The ctcMakePredictiveCall routine allows a virtual party on a switch to initiate calls on behalf of a user or group of users. Depending on your particular switch, at some time during the progress of the call, the call is allocated to a physical device. Only when the called device answers (or, for example, the phone rings a preconfigured number of times) does the call get put through to the user.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

calledNumber

type: **ctcDeviceString**
access: **read only**
mechanism: **by reference**

This argument is the address of a character string that contains the number of the device you want to call. The ASCII string can contain any combination of numbers 0 through 9 and the characters * and #.

The maximum length for calledNumber is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

allocation

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This 32-bit integer contains a value that you specify to indicate when the call is successful (for example, when the called device answers, or when the phone rings a set number of times).

To use the switch's default processing for the call, specify the value ctcK_AllocDefault. For details of other values that you can specify, refer to the switch-specific appendixes.

applicationData

type: **ctcApplString**
access: **read only**
mechanism: **by value**

You use this argument to associate data, for example, customer reference information or account details, with the call. The argument is the address of a NUL-terminated character string.

If the call is successful, the data is stored by the switch and reported on subsequent events until the call is terminated.

If you do not want to associate data with the call, pass a zero-length string.

callRefId

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This argument is the address of a 32-bit integer that receives the call reference identifier for the new call.

numberOfRings

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer specifies the number of times the destination device rings before the call fails. For details of the range of values you can pass with this argument, refer to the switch-specific appendixes.

Specify the value zero for the default number as defined on the switch.

ctcPickupCall
Pick Up a Call from Another Extension

Format in C

unsigned int ***ctcPickupCall*** *(ctcChanId channel,*
unsigned int callRefId,
ctcDeviceString calledNumber)

Description

The ctcPickupCall routine allows you to answer a call on an extension other than the one on which it is ringing.

If the ringing extension is in the same pickup group (as defined within the switch), this routine does not require the calledNumber argument specifying the dialable number for the ringing device.

If the ringing extension is not in the pickup group, you have to specify the dialable number for the ringing device (Directed Call Pickup).

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

callRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This argument is a 32-bit integer that contains a call reference identifier for the ringing call. The call reference identifier for the ringing call is returned by ctcGetEvent or ctcWinGetEvent.

calledNumber

type: **ctcDeviceString**
access: **read only**
mechanism: **by reference**

This argument is the address of a character string that identifies the ringing device. The value contained in the character string can be 0 to indicate that the call is being picked up from the local group. The maximum length for calledNumber is specified by the literal ctcMaxDnLen, in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

ctcReconnectHeld
Reconnect a Call on Hold

Format in C

*unsigned int **ctcReconnectHeld** (ctcChanId channel,*
unsigned int heldCallRefId,
unsigned int activeCallRefId)

Description

The ctcReconnectHeld routine disconnects a consultation call and reconnects a held call. It has the same effect as using both ctcCancelCall and ctcRetrieveHeld. The following example sequence shows how to use ctcReconnectHeld:

1. B calls A, and A answers.
2. A calls C using ctcConsultationCall which automatically places B on hold.
3. A uses ctcReconnectHeld to end the call to C and reconnect the call with B.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

heldCallRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the call reference identifier for the held call, as returned by ctcGetEvent or ctcWinGetEvent.

activeCallRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the call identifier value for the active call you are disconnecting.

ctcRemoveMonitor
Removes a Device From a Monitor Channel

Format in C

unsigned int ***ctcRemoveMonitor*** *(ctcChanId monitorChannel,*
ctcDeviceString deviceDN)

Description

The ctcRemoveMonitor routine removes monitoring for a device associated with a monitor channel. Use this routine when you no longer want to receive event information for the device on the monitor channel.

To stop monitoring **all** devices on a monitor channel and deassign the monitor channel, use ctcDeassign.

Restriction

This routine is not supported on CTC clients running Windows 3.1/3.11. CTC applications running on Windows 3.1/3.11 cannot assign to monitor channels.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

deviceDN

type: **ctcDeviceString**
access: **read only**
mechanism: **by reference**

This argument is the address of a character string that contains the DN for the device you no longer want to monitor:

- For a telephony device or route point, this ASCII string can contain any combination of numbers 0 through 9 and the characters * and #.

- For a monitor channel, specify the device number returned in the setDN field of the ctcChanData structure. This is returned when you call ctcGetChannelInformation for the monitor channel. See the description of ctcGetChannelInformation for more information.

Specify the device number exactly as it is returned in the setDN field, using the same case for letters. The device number is an ASCII string that can contain any combination of numbers 0 through 9, uppercase letters A through F, and the characters * and #.

The maximum length for deviceDN is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

ctcRespondToInactiveCall
Respond to Inactive Call

Format in C

*unsigned int **ctcRespondToInactiveCall*** *(ctcChanId channel,*
unsigned int callRefId,
unsigned int action)

Description

The ctcRespondToInactiveCall routine allows you to respond to an outbound call that was not completed. The call must have been made to an extension on the same switch (or possibly on a private switch network), and either the call was not answered, or the destination was busy.

This routine lets you respond by invoking one of the following functions:

- Camp on—if the destination is busy, you can wait until that call is cleared. Camp On is a mechanism for queuing calls. If you invoke Camp On, the switch notifies the destination of the presence of the queued call. When the destination device finishes its current call, and you are first in the queue, you are automatically connected.
- Barge In (also called Intrude)—you may “barge in” on the existing call, if this class of service is enabled on the switch.
- Ring Back— if the destination is busy and you invoke Ring Back, the switch will ring you back when the extension becomes free. If the extension is not answered and you invoke Ring Back, the switch will ring you back after the extension is next used.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

callRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This argument is a 32-bit integer that contains the call identifier value for the current call. The call identifier value is returned by ctcGetEvent or ctcWinGetEvent for the failed call.

action

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This argument is a 32-bit integer containing a response value. This value specifies the action required when the destination device is busy or not answering. Table 2–13 explains the meaning and effect of each possible response value.

Table 2–13 Response Values for ctcRespondToInactiveCall

Signal	Response Value	Description
Busy	ctcK_CampOn	Camp On puts the call into a special waiting state. The calling device must remain off-hook; that is, the caller simply waits until the call gets through. When the destination device becomes idle, the switch automatically rings the destination device and connects the call when it is answered.
Busy	ctcK_BargeIn	Barge In lets the caller break into a call in progress on the destination device.
Busy	ctcK_RingBack	Ring Back When Free lets the caller hang up as soon as Ring Back When Free is set. The caller can then make or receive other calls. When both parties to the original call are next free at the same time, the switch automatically rings and connects both parties.
No answer	ctcK_RingBack	Ring Back When Next Used operates as for Busy but the switch waits for an event on the destination device to indicate that there is someone there to answer the call.

ctcRespondToRouteQuery
Respond to Route Query Messages from the Switch

Format in C

unsigned int ***ctcRespondToRouteQuery*** *(ctcChanId channel,*
unsigned int routeId,
ctcDeviceString newCalledNumber,
ctcApplString applicationData)

Description

The ctcRespondToRouteQuery routine supplies a new route for a call that was presented to the application for routing by the routine ctcGetRouteQuery. See the description of the ctcGetRouteQuery routine for more information.

If you do not want the application to reroute a call, specify the address of a zero-length character string with the newCalledNumber argument.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

routeId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the route identifier returned by ctcGetRouteQuery or ctcWinGetRouteQuery for the call to be routed.

newCalledNumber

type: **ctcDeviceString**
access: **read only**
mechanism: **by reference**

This argument is the address of a null-terminated character string that identifies the new route for the call.

The maximum length for newCalledNumber is specified by the literal ctcMaxDnLen, in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

If you do not want the application to reroute a call, specify the address of a zero-length character string with this argument.

applicationData

type: **ctcApplString**
access: **read only**
mechanism: **by value**

You use this argument to associate data, for example, customer reference information or account data, with the call being routed. The argument specifies the address of a NUL-terminated character string.

If the call is successfully routed, the data is stored by the switch and reported on subsequent events until the call is terminated.

Note that if data is already associated with the call, it is overwritten by the string that you specify. If you do not want to overwrite the data, or if you do not want to associate any data with the call, pass a zero-length string.

ctcRetrieveHeld
Retrieve a Call on Hold

Format in C

*unsigned int **ctcRetrieveHeld** (ctcChanId channel,*
unsigned int callRefId)

Description

The ctcRetrieveHeld routine retrieves a call that is on hold.

Some switches require you to cancel the consultation call before retrieving the call on hold. With other switches, the cancel function is not required because ctcRetrieveHeld cancels the call itself.

For example, for A to cancel a consultation call and retrieve a call on hold:

1. A calls B, and then calls C using ctcConsultationCall, which automatically puts B on consultation hold.
2. Depending on the switch, A can do one of the following:
   - Use ctcCancelCall to cancel the call to C. This routine disconnects C and puts A in the initiate state. A can now use ctcRetrieveHeld to retrieve the call to B, who is on consultation hold.
   - Use ctcRetrieveHeld to cancel the call to C and retrieve the call to B.

If you require your application to work with all switches that CTC supports, always attempt to cancel the failed consultation call with the ctcCancelCall routine. If this routine returns with ctcUnsupProc, use the ctcRetrieveHeld routine to cancel the consultation call and return to the held call.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier channel ID) value returned by ctcAssign for the device in use.

callRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the call identifier value for the held call you wish to retrieve. This value is returned by ctcGetEvent or ctcWinGetEvent.

ctcSendDTMF
Output DTMF Tones

Format in C

*unsigned int **ctcSendDTMF** (ctcChanId channel,*
unsigned int callRefId,
ctcDeviceString DTMFdigits)

Description

The ctcSendDTMF routine generates DTMF tones over the telephone line. DTMF tones are usually generated by pressing the keys on a telephone keypad. This routine enables an agent to respond to systems that require DTMF tones as input.

You specify the tones you want to generate with the calledNumber argument. This argument specifies the string of numbers you want to convert for output as DTMF tones.

Restrictions

To use ctcSendDTMF, both of the following must apply:

- The channel must be assigned to a voice set (for example, a telephone) or an agent position. This routine is not supported for channels assigned to logical devices (call queues, route points, or monitor channels).
- There must be an active call on the line.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

callRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the call identifier value for the current call.

The call identifier value is the latest value returned by ctcGetEvent or ctcWinGetEvent for the call.

DTMFdigits

type: **ctcDeviceString**
access: **read only**
mechanism: **by reference**

This argument is the address of a character string that specifies the numbers you want to convert for output as DTMF tones.

The string can contain any combination of the numbers 0 through 9 and the characters * and #, up to a maximum of characters specified by the literal ctcMaxDnLen. This literal is defined in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

ctcSetAgentStatus
Set Status for an ACD Agent

Format in C

unsigned int ***ctcSetAgentStatus****(ctcChanId channel,*
unsigned int agentMode,
ctcDeviceString agentData,
ctcDeviceString logicalAgent,
ctcDeviceString agentGroup)

Description

The ctcSetAgentStatus routine lets you set the status for an ACD agent. Using this routine, a user can log on (with an optional password) or log off as an ACD agent. They can also declare themselves:

- Ready to take calls
- Busy
- Completing details after a call
- Doing other work

You can also use the agentGroup argument to associate the agent with a specific agent group, if, for example, the switch enables agents to log into more than one agent group.

Arguments

channel

type:	**ctcChanId**
access:	**read only**
mechanism:	**by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

agentMode

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This argument specifies the status for an ACD agent. It contains one of the values in Table 2–14.

Table 2–14 Agent Mode Values for ctcSetAgentStatus

Value	Description
ctcK_AgentReady	The agent is ready to receive calls
ctcK_AgentNotReady	The agent is not ready to receive calls
ctcK_AgentOtherWork	The agent is involved in other work and cannot take calls
ctcK_AgentAfterCallWork	The agent is completing details of a call
ctcK_AgentLogin	The agent is logging in
ctcK_AgentLogout	The agent is logging out

agentData

type: **ctcDeviceString**
access: **read only**
mechanism: **by reference**

This argument contains optional data (such as a password).

The maximum length for agentData is specified by the literal ctcMaxDnLen in the CTC definitions file. Note that this maximum length includes the null termination character (NUL).

logicalAgent

type: **ctcDeviceString**
access: **read only**
mechanism: **by reference**

This argument contains optional data for logical agents. Use this argument to specify the DN (for example, telephone number) for the logical agent.

The maximum length for logicalAgent is specified by the literal ctcMaxDnLen in the CTC definitions file. Note that this maximum length includes the null termination character (NUL).

ctcSetAgentStatus

agentGroup

type: **ctcDeviceString**
access: **read only**
mechanism: **by reference**

This argument contains optional data. It specifies the DN for an agent group. If your switch enables agents to log into more than one agent group, use this argument to specify with which group the agent is associated.

The maximum length for agentGroup is specified by the literal ctcMaxDnLen in the CTC definitions file. Note that this maximum length includes the null termination character (NUL).

ctcSetCallForward
Set Call Forward for a Device

Format in C

unsigned int ***ctcSetCallForward*** *(ctcChanId channel,*
unsigned int forwardMode,
ctcDeviceString forwardDn)

Description

The ctcSetCallForward routine lets you set call forward for the assigned device so that incoming calls are redirected to another device. You can use this routine to set the following:

- All calls to be forwarded
- External calls only to be forwarded
- Internal calls only to be forwarded

You can also specify whether incoming calls are to be forwarded if the assigned device is busy or if the call is not answered after a period of time (determined by the switch).

Canceling Call Forward

You also use ctcSetCallForward to cancel call forward. Use the forwardMode argument with the same value that you specified to set call forward on, but do not specify a value for forwardDN; that is, specify the address of a zero-length character string.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

forwardMode

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This argument sets the call forward setting for the assigned device. It contains one of the values in Table 2–15.

Table 2–15 Call Forward Values for ctcSetCallForward

Value	Calls Forwarded
ctcK_CfAll	All calls
ctcK_CfExtBusy	External calls when the assigned device is busy
ctcK_CfExtNoAnswer	External calls when there is no answer
ctcK_CfIntBusy	Internal calls when the assigned device is busy
ctcK_CfIntNoAnswer	Internal calls when there is no answer
ctcK_CfNoAnswerBusy	All calls when there is no answer and when the assigned device is busy

forwardDN

type: **ctcDeviceString**
access: **read only**
mechanism: **by reference**

This argument is the address of a character string that contains the dialable number of the destination device.

The maximum length for forwardDN is specified by the literal ctcMaxDnLen, in the CTC definitions file. Note that this maximum length includes the null termination character (NUL).

ctcSetDoNotDisturb
Set Do-Not-Disturb for a Device

Format in C

unsigned int ***ctcSetDoNotDisturb*** *(ctcChanId channel,*
unsigned int DNDMode)

Description

The ctcSetDoNotDisturb routine sets or cancels Do-Not-Disturb for the assigned device. When Do-Not-Disturb is set on, incoming calls do not ring at the device. Your switch-specific documentation should describe what happens to a call when it encounters a Do-Not-Disturb feature.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

DNDMode

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains one of the values in the following table:

Value	Description
ctcK_On	Sets Do-Not-Disturb on for the assigned device
ctcK_Off	Sets Do-Not-Disturb off for the assigned device

When Do-Not-Disturb is set on, incoming calls are not presented at the assigned device. The default setting is off.

ctcSetMessageWaiting
Set Message Waiting for a Device

Format in C

unsigned int ***ctcSetMessageWaiting*** *(ctcChanId channel,*
unsigned int messageWaitingMode)

Description

The ctcSetMessageWaiting routine lets you set the message waiting indicator on or off for the assigned device. The message waiting indicator is usually a lamp on the telephone set. If there is a message waiting, the lamp is lit.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

messageWaitingMode

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains one of the values in the following table:

Value	Description
ctcK_On	Sets the message waiting indicator on
ctcK_Off	Sets the message waiting indicator off

The indicator is often a lamp on the device.

ctcSetMonitor
Set Monitoring for a Device

Format in C

unsigned int ***ctcSetMonitor*** *(ctcChanId channel,*
unsigned int monitorMode)

Description

The ctcSetMonitor routine changes the monitoring state of the assigned device.

You can use this routine with ctcGetEvent to receive useful information on the state of calls associated with a device. Status information is returned whenever a significant event occurs; for example, when an incoming call arrives, or when an active call is disconnected.

Monitoring Devices

Monitoring a device can provide information on the other party or parties involved in a phone call. It can return:

- The extension numbers for those parties on the same switch
- For an outside call, the trunk number in use on the switch or, if the switch and telephone network can supply the information, the calling line ID, which identifies the device that originated the call
- The dialed number (the digits used to place the call)

Monitoring also returns a reference number for calls on the assigned device. This call reference identifies the call, and you must use it as a parameter for most CTC routines that process telephone calls.

Monitoring Groups or Call Queues

CTC can return information when a call enters or leaves a specific group queue, and can tell you if the caller has disconnected or if the call has been routed to an agent.

ctcSetMonitor

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

monitorMode

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This argument is a 32-bit integer that contains one of the values in the following table:

Value	Description
ctcK_On	Sets monitoring on for the device
ctcK_Off	Sets monitoring off for the device

ctcSetRoutingEnable
Set the Routing State for a Device

Format in C

*unsigned int **ctcSetRoutingEnable** (ctcChanId channel,*
unsigned int routingMode)

Description

The ctcSetRoutingEnable routine enables or disables routing for the assigned route point:

- If you enable routing, the switch passes a route request to CTC when it receives a call for the assigned route point. You use ctcGetRouteQuery and ctcWinGetRouteQuery to receive the route request and ctcRespondToRouteQuery to provide a new route for the call.
- If you disable routing, the switch stops sending route requests to CTC for the assigned route point.

To display the current routing state for a route point, use ctcGetRoutingEnable. Refer to the description of ctcGetRoutingEnable for more information.

Enabling Call Routing

If your switch supports ctcSetRoutingEnable, you use the following sequence of routines:

1. ctcAssign to assign a channel to the route point.
2. ctcSetRoutingEnable to explicitly enable call routing for the assigned route point. The switch passes route requests to CTC whenever it receives a call at the route point.
3. ctcGetRouteQuery or ctcWinGetRouteQuery to receive the route requests.
4. ctcRespondToRouteQuery to respond to the route requests.

At any point, you can check whether routing is enabled or disabled for a route point with the ctcGetRoutingEnable routine.

For more information, refer to the descriptions of the ctcGetRoutingEnable, ctcGetRouteQuery, ctcWinGetRouteQuery, and ctcRespondToRouteQuery routines.

Disabling Call Routing

When you disable call routing, the switch stops sending route requests to your application for calls made to the route point. If there is an outstanding ctcGetRouteQuery or ctcWinGetRouteQuery request, an error is returned.

You can continue to receive information about calls made to the route point by using ctcGetEvent. If supported by your switch, you can continue to monitor the route point with this routine.

Using Call Routing Without ctcSetRoutingEnable

A number of switches support call routing but pass routing requests to CTC automatically. On these switches, you cannot enable or disable routing for a specific route point so ctcSetRoutingEnable is not supported. However, you can continue to receive and respond to route requests by using the following sequence of routines:

1. ctcAssign to assign a channel to the route point
2. ctcGetRouteQuery or ctcWinGetRouteQuery to receive the route request
3. ctcRespondToRouteQuery to provide a new route for the call

To check whether your switch supports ctcSetRoutingEnable, refer to Appendix A. Note that if your switch supports ctcSetRoutingEnable, you must use this command to explicitly enable routing before you use ctcGetRouteQuery or ctcWinGetRouteQuery.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype and identifies the route point for which you want to receive route requests.

Specify the channel identifier returned by ctcAssign for the route point.

routingMode
type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This argument is a 32-bit integer that contains one of the values in the following table:

Value	Description
ctcK_On	Enables routing for the assigned route point
ctcK_Off	Disables routing for the assigned route point

ctcSingleStepTransfer
Make a Call Transfer

Format in C

```
unsigned int ctcSingleStepTransfer (ctcChanId         channel,
                                    ctcDeviceString   calledNumber,
                                    unsigned int      callRefId,
                                    ctcApplString     applicationData,
                                    unsigned int      *newCallRefId)
```

Description

The ctcSingleStepTransfer routine transfers a current call to a third party and disconnects the assigned device.

ctcSingleStepTransfer enables you to complete an unscreened (or unsupervised) transfer without first placing the current call on hold.

For example, for A to transfer to C an incoming call from B:

1. B calls A using ctcMakeCall, and A answers.
2. A uses ctcSingleStepTransfer to put the call through to C. A is automatically disconnected and B waits for C to answer.

For screened transfer (for example, for A to wait until C answers before transferring the call), use ctcConsultationCall and ctcTransferCall.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

calledNumber

type: **ctcDeviceString**
access: **read only**
mechanism: **by value**

This character string contains the number of the device to which you are transferring the call. The ASCII string can contain any combination of numbers 0 through 9 and the characters * and #.

The maximum length for calledNumber is specified by the literal ctcMaxDnLen in the CTC definitions file. Note that this maximum length includes the null termination character (NUL).

callRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the call identifier value for the call you want to transfer. Specify the latest value returned by ctcGetEvent, ctcWinGetEvent, or an associated routine such as ctcMakeCall.

applicationData

type: **ctcApplString**
access: **read only**
mechanism: **by value**

This argument is the address of a NUL-terminated character string that you want to associate with the call to be transferred. For example, customer reference information or account data.

If the call transfer is successful, the data is stored by the switch and reported on subsequent events until the call is terminated.

If you do not want to associate data with the call, pass a zero-length string.

newCallRefId

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This argument is the address of a 32-bit integer into which CTC writes a call identifier value for the new transferred call.

ctcSnapshot
Query the Current State of a Device

Format in C

unsigned int ***ctcSnapshot*** *(ctcChanId channel,*
*ctcCallData *callData,*
*unsigned int *numberOfCalls)*

Description

The ctcSnapshot routine returns information for up to 32 current calls at the assigned telephony device. ctcSnapshot provides the following information:

- Call reference for each call
- State of each call
- Total number of calls at the device or in the queue

For example, if the user at a voice set places a call on hold and then makes a consultation call, ctcSnapshot returns:

- A call reference and state for the held call
- A call reference and state for the consultation call
- The total number of calls at the voice set (2)

For queues, ctcSnapshot can return call references and states for up to 32 calls. If there are more than 32 calls in the queue, CTC returns the first 32 call references and states provided by the switch. The numberOfCalls argument returns the total number of calls in the queue.

Restrictions

ctcSnapshot is supported for channels assigned to devices of type ctcK_Dn only. For example, voice sets or queues. For more information, refer to the description of ctcAssign.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

callData

type: **ctcCallData**
access: **write only**
mechanism: **by reference**

This argument contains the address of an array of 32 fixed-format structures, of type ctcCallData and formatted as follows:

```
ctcCallData{
   unsigned int     refId;
   unsigned int     state;
 };
```

The fields in this structure are all null terminated.

refId

This 32-bit field returns the reference identifier for a call.

state

This 32-bit field returns the state for a call.

For details of the call states that can be returned, refer to Table 2–4.

numberOfCalls

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This 32-bit integer returns the total number of calls at the assigned device.

ctcSwapWithHeld
Swap with Call on Hold

Format in C

unsigned int ***ctcSwapWithHeld*** *(ctcChanId channel,*
unsigned int heldCallRefId,
unsigned int activeCallRefId)

Description

The ctcSwapWithHeld routine swaps the current call with the call on consultation hold.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

heldCallRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains a call identifier value for the held call you wish to swap.

activeCallRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains a call identifier value for the active call you wish to swap (with the call on hold).

ctcTransferCall
Transfer a Call

Format in C

unsigned int ***ctcTransferCall*** *(ctcChanId channel,*
unsigned int heldCallRefId,
unsigned int activeCallRefId,
*unsigned int *newCallRefId)*

Description

The ctcTransferCall routine completes the transfer of a call (initiated by the ctcConsultationCall routine) to a different extension, and disconnects the assigned device.

For example, for A to transfer to C an incoming call from B (where A's current call is the call from B):

1. B calls A, using ctcMakeCall, and A answers.
2. A calls C, using ctcConsultationCall, which automatically puts the call from B on hold.
3. A invokes ctcTransferCall when connected to C. B and C are now connected and A is disconnected.

To screen (or supervise) a transfer, A waits until speaking to C before invoking ctcTransferCall. For unscreened (or unsupervised) transfer, A invokes ctcTransferCall before C answers the telephone.

Arguments

channel

type:	**ctcChanId**
access:	**read only**
mechanism:	**by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

ctcTransferCall

heldCallRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the call identifier value for the held call to be transferred.

The call identifier value is the latest value returned by ctcWinGetEvent for the held call.

activeCallRefId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer contains the call identifier value for the active call to which you want to transfer the held call.

The call identifier value is the latest value returned by ctcWinGetEvent for the active call.

newCallRefId

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This argument is the address of a 32-bit integer that receives a call identifier value for the new transferred call.

ctcWinGetEvent
Get Information About Event and State Changes

Format in C

unsigned int ***ctcWinGetEvent*** *(ctcChanId channel,*
*lpvASB *lpvASB,*
HWND hWnd,
*ctcEventData *eventData)*

Description

ctcWinGetEvent is a non-blocking routine that enables a Windows-based CTC application to receive telephony events for the assigned device. It returns the same information as ctcGetEvent.

CTC clients running Windows 3.1/3.11 must use ctcWinGetEvent, not ctcGetEvent, to receive event information. ctcWinGetEvent is an asynchronous routine that enables Windows 3.1/3.11 clients to receive event information for the assigned device without blocking the application. Windows 3.1/3.11 clients require a non-blocking routine because they use a Windows Socket interface to the CTC server for API calls and not DCE Remote Procedure Call services.

Note that ctcWinGetEvent is available for Windows 3.1/3.11 clients only. For Windows NT and Windows 95 clients, use ctcGetEvent.

How ctcWinGetEvent Returns Event Data

When an event occurs at the assigned device, CTC:

- Returns the event in the ctcEventData structure
- Posts a PM_CTC_EVENT completion message to the window specified by the hWnd argument

Associated with the completion message is an lParam parameter that specifies the address for the lpvASB structure. The lpvASB structure contains the routine completion status and a read-only value, for example, a pointer to the ctcEventData structure into which event information has been written.

The amount of information that CTC returns depends on the information provided by the switch. This may be different for a call that is internal to the switch and for an outside call, depending on the type of trunks connected to the switch.

ctcWinGetEvent

Calling ctcWinGetEvent

To return event information for the assigned device, you use the following sequence of routines:

1. ctcSetMonitor to set monitoring on
2. ctcWinGetEvent

Note that you do not need to call ctcWinGetEvent after each event. You only need to call this routine again if the completion status returned in the lpbASB structure is a value other than ctcSuccess or ctcEventDataLost.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

lpvASB

type: **lpvASB**
access: **write only**
mechanism: **by reference**

This argument contains the address of a fixed-format structure, for which you allocate memory of type lpvASB. The structure is defined in a CTC definitions file installed on your system.

The lpvASB structure is formatted as follows:

```
lpvASB{
   unsigned int     dwStatus;
   unsigned int     lpvDataPointer;
   unsigned int     lpvChannel;
 };
```

The following information is returned in the lpvASB structure:

- **dwStatus**

  On completion of the asynchronous procedure, this field contains the routine completion status. This is a write only value.

- **lpvDataPointer**

  This field contains a read only value. For example, if you are monitoring multiple channels, you can use this field to identify the ctcEventData structure into which event information has been written.

- **lpvChannel**

  On completion of the asynchronous procedure, this field contains the identifier for the channel for which you want event information.

hWnd

type: **HWND**
access: **read only**
mechanism: **by reference**

This handle specifies the window where CTC returns the PM_CTC_EVENT message generated by an event at the assigned device.

eventData

type: **ctcEventData**
access: **write only**
mechanism: **by reference**

This argument contains the address of a fixed-format structure, ctcEventData. This is shown on the following page.

For a description of the fields in the ctcEventData structure, refer to the description of the ctcGetEvent routine.

ctcWinGetEvent

```
ctcEventData{
   unsigned int      refId;
   unsigned int      netCallId;
   unsigned int      oldRefId;
   unsigned int      oldNetCallId;
   unsigned int      state;
   unsigned int      event;
   unsigned int      eventQualifier;
   unsigned int      type;
   unsigned int      otherPartyType;
   unsigned int      otherPartyQualifier;
   ctcDeviceString  otherParty;
   unsigned int      otherPartyTrunk;
   unsigned int      otherPartyGroup;
   unsigned int      thirdPartyType;
   unsigned int      thirdPartyQualifier;
   ctcDeviceString  thirdParty;
   unsigned int      thirdPartyTrunk;
   unsigned int      thirdPartyGroup;
   unsigned int      calledPartyType;
   unsigned int      calledPartyQualifier;
   ctcDeviceString  calledParty;
   unsigned int      calledPartyTrunk;
   unsigned int      calledPartyGroup;
   ctcApplString    applicationData;
   ctcDeviceString  monitorParty;
   ctcDeviceString  nestedMonitorChannel;
   unsigned int      agentMode;
   ctcDeviceString  agentId;
   ctcDeviceString  agentGroup;
   ctcDeviceString  agentData;
   ctcDeviceString  logicalAgent;
   ctcDeviceString  dtmfDigits;
   unsigned int      originatingPartyType;
   unsigned int      originatingPartyQualifier;
   ctcDeviceString  originatingParty;
   unsigned int      originatingPartyTrunk;
   unsigned int      originatingPartyGroup;
   unsigned int      secOldRefId;
   unsigned int      callsQueued;
   ctcAccountInfo   accountInfo;
   ctcTimeStamp     timeStamp;
   unsigned int      privateData;
 };
```

ctcWinGetRouteQuery
Get Route Query Messages from the Switch

Format in C

unsigned int ***ctcWinGetRouteQuery*** *(ctcChanId*	*channel,*
	lpvASB	**lpvASB,*
	HWND	*hWnd,*
	ctcRouteData	**routeData)*

Description

ctcWinGetRouteQuery is a non-blocking routine that presents a call to a Windows-based application so that the call can be routed. If supplied by the switch, this routine can also return information on the other parties involved in the call and the number dialed by the caller.

CTC clients running Windows 3.1/3.11 must use ctcWinGetRouteQuery for call routing. ctcWinGetRouteQuery is an asynchronous routine that enables Windows 3.1/3.11 clients to receive route information for the assigned route point without blocking the application. Windows 3.1/3.11 clients require a non-blocking routine because they use a Windows Socket interface to the CTC server for API calls and not DCE Remote Procedure Call services.

ctcWinGetRouteQuery returns the same information as ctcGetRouteQuery.

Note that ctcWinGetRouteQuery is available for Windows 3.1/3.11 clients only. For Windows NT and Windows 95 clients, use ctcGetRouteQuery.

How ctcWinGetRouteQuery Returns Route Data

When a route request occurs at the assigned device, CTC:

- Returns the information in the ctcRouteData structure
- Posts a PM_CTC_ROUTE completion message to the window specified by the hWnd argument

Associated with the completion message is an lParam parameter that specifies the address for the lpvASB structure. The lpvASB structure contains the routine completion status and a read only value, for example, a pointer to the ctcRouteData structure into which route information has been written.

ctcWinGetRouteQuery

How to Call ctcWinGetRouteQuery

Use the following sequence of routines:

1. ctcAssign to assign the channel to a route point acting as a device. The route point has a DN associated with it, but there is no real, physical device.
2. ctcWinGetRouteQuery to be notified of route requests for calls to the assigned route point.
3. ctcRespondToRouteQuery to supply a route for the call.

Note that you do not need to call ctcWinGetRouteQuery after receiving notification of a route request. You only need to call this routine again if the completion status returned in the lpbASB structure is a value other than ctcSuccess or ctcEventDataLost.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

lpvASB

type: **lpvASB**
access: **write only**
mechanism: **by reference**

This argument contains the address of a fixed-format structure, for which you allocate memory of type lpvASB. The structure is defined in a definitions file installed on your system.

The lpvASB structure is formatted as follows:

```
lpvASB{
   unsigned int     dwStatus;
   unsigned int     lpvDataPointer;
   unsigned int     lpvChannel;
 };
```

The following information is returned in the lpvASB structure:

- **dwStatus**

  On completion of the asynchronous procedure, this field contains the routine completion status. This is a write only value.

- **lpvDataPointer**

  This field contains a read only value. For example, if you are monitoring multiple channels, you can use this field to identify the ctcRouteData structure into which route information has been written.

- **lpvChannel**

  On completion of the asynchronous procedure, this field contains the identifier for the channel for which you want route information.

hWnd

type: **HWND**
access: **read only**
mechanism: **by reference**

This handle specifies the window where CTC returns the PM_CTC_ROUTE message generated by a route request at the assigned route point.

routeData

type: **ctcRouteData**
access: **write only**
mechanism: **by reference**

This argument contains the address of a fixed-format structure, for which you allocate memory of type ctcRouteData. The structure is defined in a CTC definitions file (see Section 1.5) and is formatted as shown on the following page.

CTC returns information in one or more of the ctcRouteData fields. These are described in Table 2–16.

```
ctcRouteData{
   unsigned int     routeId;
   unsigned int     refId;
   unsigned int     spare001;
   unsigned int     otherPartyType;
   ctcDeviceString  otherParty;
   unsigned int     otherPartyTrunk;
   unsigned int     otherPartyGroup;
   unsigned int     thirdPartyType;
   ctcDeviceString  thirdParty;
   unsigned int     thirdPartyTrunk;
   unsigned int     thirdPartyGroup;
   unsigned int     calledPartyType;
   ctcDeviceString  calledParty;
   unsigned int     calledPartyTrunk;
   unsigned int     calledPartyGroup;
   ctcApplString    applicationData;
   ctcDeviceString  dtmfDigits;
   ctcTimeStamp     timeStamp;
   unsigned int     privateData;
 };
```

Table 2–16 Information Returned by ctcWinGetRouteQuery

Field	Contents
routeId	The route identifier for the call to be routed. Use this identifier to specify a new route for the call with ctcRespondToRouteQuery.
refId	The call reference returned by ctcWinGetEvent.
spare001	Null data.
otherPartyType	A value that identifies the number of the other party as a CLID, DN, or DNIS.
otherParty	The CLID, DN, or DNIS for the other party.
otherPartyTrunk	A trunk line number from the switch for the other party.
otherPartyGroup	A trunk group number from the switch for the other party.
thirdPartyType	A value that identifies the number of an additional party on the call as a CLID, DN, or DNIS.
thirdParty	The CLID, DN, or DNIS of an additional party involved in the call.
thirdPartyTrunk	A trunk line number from the switch for the third party.
thirdPartyGroup	A trunk group number from the switch for the third party.
calledPartyType	A value that identifies the number originally dialed as a CLID, DN, or DNIS.
calledParty	The CLID, DN, or DNIS originally dialed.
calledPartyTrunk	A trunk line number from the switch for the called party.

Table 2–16 Information Returned by ctcWinGetRouteQuery (Continued)

Field	Contents
calledPartyGroup	A trunk group number from the switch for the called party.
applicationData	Data associated with the call to be routed.
dtmfDigits	DTMF digits collected as the call is routed.
timeStamp	Time the route request was processed either by the CTC server or by the switch (depending on the configuration of the link).
privateData	A value to indicate whether CTC has received private data from the switch.

For detailed information about the values returned in these fields, refer to the description of ctcGetEvent. For details of the fields supported by your switch, refer to the switch-specific appendixes.

ctcWinGetRouteQuery

3

Errors and Conditions Returned

Table 3–1 in this chapter lists by name the errors and conditions that can be returned by CTC routines. It also provides a brief description of each error and condition. Use Table 3–1 in conjunction with the ctcErrMsg routine. This routine provides the name of the condition or error associated with a returned value. For details of ctcErrMsg, refer to Chapter 2.

3.1 Mapping Errors to Routines

It is not possible to specify which specific errors and conditions can be returned for each CTC routine. Different errors and conditions can be returned by different switches for the same routine.

However, to help you determine and isolate problems, Table 3–1 also indicates the source of the condition the CTC API, CTC server, or the switch and the type of error returned for conditions reported by the switch.

3.2 Source of Errors

The following general guidelines apply:

- Errors from the CTC API are usually returned for programming errors. For example, if you pass in an invalid type or argument.
- Condition values from the CTC server are usually associated with resources or CTC management.
- Condition values from the switch are often returned when there is a problem with the device state or the call reference. For example, when you try to perform an operation and the device is in the wrong state for that operation, or when you provide an invalid call reference.

3.3 Types of Errors Returned by the Switch

Where possible, for each condition or error returned by the switch, Table 3–1 specifies one of the following:

Error	Description
Operation Error	An error in the service request made to the switch.
State Incompatibility Error	The service request is incompatible with the condition of a related CSTA object.
System Resource Error	The service request is not fulfilled because there are insufficient system resources at the switch.
Subscribed Resource Availability Error	The service request is not fulfilled because the required switch resource is not available.
Security Error	Security error on the switch.
Performance Management Error	Performance management error on the switch.
Unspecified Error	A switch error has occurred that does not map onto the other error types.

Table 3–1 Condition Values Returned

Error	From	Description
ctcAlreadyOn	CTC SERVER	Monitoring is already set on for this channel.
ctcAsn1DecodeErr	CTC SERVER	A bad message format record was received from the switch.
ctcAsn1EncodeErr	CTC SERVER	A bad parameter was supplied in the message to the ASN1 encoding routine.
ctcAssignLimitReached	CTC SERVER	The application has assigned the maximum number of channels allowed by the CTC software license.
ctcBadObjState	SWITCH	State incompatibility error. The object is in the incorrect state for the service. The switch is unable to provide more specific information.
ctcBindFail	CTC API	An RPC network binding handle cannot be created from the serverName and networkType arguments for ctcAssign.
ctcComFail	CTC SERVER	Insufficient virtual memory has been detected during the communications initialization procedure.
ctcCondError	CTC SERVER	An internal error has occurred on the CTC server.
ctcCondWaiting	CTC SERVER	An internal error has occurred on the CTC server.
ctcConfMemberLimEx	SWITCH	System resource error. The request exceeded the switch's limit for the number of members of a conference.
ctcDeadLock	CTC SERVER	An internal error has occurred on the CTC server.
ctcEventDataLost	CTC SERVER	A number of events have occurred at the same time and some event data has been lost.
ctcEventInProgress	CTC SERVER	A previous ctcGetEvent or ctcWinGetEvent request has not yet completed.
ctcExTrunkLimExc	SWITCH	Subscribed Resource Availability Error. The request exceeded the limit for external trunks as defined on the switch.

Table 3–1 Condition Values Returned (Continued)

Error	From	Description
ctcFileOpenError	CTC SERVER	The CTC trace facility could not create the specified trace file. Check that the file specification is correct and that disk space is available. Refer to the *CT-Connect Installation and Administration Guide* for details of the CTC Control Program TRACE command.
ctcInitFail	CTC SERVER	Insufficient virtual memory has been detected during the communications initialization procedure.
ctcInsMem	CTC SERVER	There is insufficient virtual memory available on the CTC server to complete the requested CTC function. Ask your system manager to increase the amount available and restart the CTC server.
ctcInternErr	CTC SERVER	An unspecified internal error has occurred on the CTC server. Report the problem to Dialogic.
ctcInvAccountCode	SWITCH	An invalid account code was specified. This value is returned by CSTA Phase II switches only.
ctcInvAgentData	CTC API	Operation error. The agentData argument for ctcSetAgentStatus contains an invalid value for example, an invalid password.
ctcInvAgentMode	CTC API	The agentMode argument for ctcSetAgentStatus contains an invalid value.
ctcInvalidDest	SWITCH	Operation error. The service request specified an invalid destination.
ctcInvalidFeature	SWITCH	Operation error. The service request specified an invalid feature.
ctcInvAllocState	SWITCH	Operation error. The service request specified an invalid allocation condition.
ctcInvApplCorrelator	SWITCH	The CSTA application correlator data parameter is not valid. This error is returned by CSTA Phase II switches only.
ctcInvAuthCode	SWITCH	An invalid authorization code was specified. This condition value is returned by CSTA Phase II switches only.
ctcInvCalledDevice	SWITCH	Operation error. The called device is invalid.

Table 3–1 Condition Values Returned (Continued)

Error	From	Description
ctcInvCallFwdMode	CTC API	The forwardMode argument for ctcSetCallForward contains an invalid value.
ctcInvCallIdentifier	SWITCH	Operation error. The call identifier is invalid.
ctcInvCallingDevice	SWITCH	Operation error. The calling device is invalid.
ctcInvChannel	CTC API	The specified channel identifier is not valid. Check that you specified the channel identifier as returned by ctcAssign for the device in use.
ctcInvConnIdActCall	SWITCH	State incompatibility error. The call identifier specified in the Active Call parameter of the request is invalid.
ctcInvConnIdentifier	SWITCH	Operation error. The CSTA connection identifier is invalid.
ctcInvCrossRefId	SWITCH	Operation error. The service request specified a cross reference identifier that is not in use at this time.
ctcInvDeviceType	CTC API	The specified device type is invalid.
ctcInvDevIdentifier	SWITCH	Operation error. The device identifier is invalid.
ctcInvDN	CTC API	The specified DN is not recognized on the switch. Specify a valid number.
ctcInvDNDMode	CTC API	The DNDMode argument for ctcSetDoNotDisturb contains an invalid value.
ctcInvForwardingDest	SWITCH	Operation error. The request cannot be provided because the forwarding destination device is invalid.
ctcInvLogID	CTC API	The logicalIdentifier argument for ctcAssign contains an invalid string.
ctcInvMonitorMode	CTC API	The monitorMode argument for ctcSetMonitor contains an invalid value.
ctcInvNetType	CTC API	The networkType argument for ctcAssign contains an invalid or unsupported RPC protocol sequence string.
ctcInvObjectType	SWITCH	Operation error. The service request specified an invalid object type for the service.

Table 3–1 Condition Values Returned (Continued)

Error	From	Description
ctcInvParam	CTC API	One of the arguments specified with a CTC routine contains an invalid value.
ctcInvPrivateData	CTC API	The private data passed was invalid. For example, the data may be too large. This value is returned by CSTA switches only.
ctcInvRouteData	CTC API	The route information specified is invalid.
ctcInvServerName	CTC API	The serverName parameter for ctcAssign contains an invalid name or address string for the CTC server.
ctcLibFail	CTC SERVER	The CTC server was unable to load the modules for the protocol specified in the CTC server startup by the Control Program SET LINK command. Report the problem to Dialogic.
ctcLinkConnectFail	CTC SERVER	The connection over the link between the CTC server and the switch has failed.
ctcLinkDown	CTC SERVER	The link between the CTC server and the switch is down.
ctcLinkInUse	CTC SERVER	There are still channels assigned over the link between the CTC server and the switch. Ensure that all users have stopped accessing the CTC server before you set the link off.
ctcLinkReset	CTC SERVER	There was an error on the link between the CTC server and the switch so the CTC server has reset the link.
ctcLinkUp	CTC SERVER	A connection has been made over the link between the CTC server and the switch.
ctcMCBnoUCB	CTC SERVER	An internal error has occurred on the CTC server.
ctcMissingParam	CTC API	A required argument was not specified. Refer to Chapter 2 for descriptions of the arguments required for the routine called.
ctcMonAlreadyOn	CTC API	Monitoring is already set on for this channel.
ctcMonCleared	CTC API	Returned for Meridian switches only. Monitoring has stopped because the Meridian Mail system released the assigned voice channel.

Table 3–1 Condition Values Returned (Continued)

Error	From	Description
ctcMonitorOff	CTC API	Monitoring is set off for this channel so the call to ctcGetEvent or ctcWinGetEvent has been returned.
ctcMonMaxExceeded	CTC SERVER	The number of channels being monitored has exceeded the maximum as defined on the switch or by CTC. Check with your switch administrator for details of the maximum number of monitors defined on your switch. Refer to the *CT-Connect Installation and Administration Guide* for details of how to use the Control Program SET MONITORS command to change the maximum set by CTC. If this error is still returned, it may indicate that you have exceeded the number of monitors allowed by your CTC software agreement.
ctcMonNotOn	CTC API	Monitoring is not set on for this channel.
ctcMutexLocked	CTC SERVER	An internal error has occurred on the CTC server.
ctcNetBusy	SWITCH	System resource error. The switch or the network is busy.
ctcNetOutOfServ	SWITCH	System resource error. The switch or network is out of service.
ctcNoActiveCall	SWITCH	State incompatibility error. The requested service operates on an active call, but there is no active call.
ctcNoCallToAnswer	SWITCH	State incompatibility error. There is no active call associated with the CTC call identifier of the call to be answered.
ctcNoCallToClear	SWITCH	State incompatibility error. There is no call associated with the CTC call identifier of the clear call request.
ctcNoCallToComplete	SWITCH	State incompatibility error. There is no active call for the CTC call identifier specified as the call to be completed.
ctcNoConnToClear	SWITCH	State incompatibility error. There is no connection associated with the CSTA connection identifier specified as the connection to be cleared.

Table 3–1 Condition Values Returned (Continued)

Error	From	Description
ctcNoEvent	CTC API	The dontWait argument for ctcGetEvent is set to TRUE and there is no event data at the CTC server for this channel.
ctcNoHeldCall	SWITCH	State incompatibility error. The requested service operates on a held call, but the specified call is not on hold.
ctcNoPrivateData	CTC API	No private data is available from the switch. This value is returned by CSTA switches only.
ctcNoRoute	CTC API	There is no route data at the CTC server for this channel.
ctcNoRouteReq	CTC API	No call was presented to the application for routing when ctcRespondToRouteQuery was called. Your application must call ctcGetRouteQuery or ctcWinGetRouteQuery before it provides a new route with ctcRespondToRouteQuery.
ctcNotOn	CTC SERVER	Monitoring is not set on for this channel.
ctcNoUnlock	CTC SERVER	An internal error has occurred on the CTC server.
ctcObjectNotKnown	SWITCH	Operation error. The parameter has a value that is not known to the switch.
ctcObjMonLimEx	SWITCH	Subscribed resource availability error. The request exceeded the switch's limit of monitors for the specified object.
ctcOpGeneric	SWITCH	Operation error. Either the switch is unable to provide more information or it has detected an undefined error.
ctcOptNotSup	CTC API	The specified request is not supported by the switch.
ctcOutstandReqLimEx	SWITCH	Subscribed resource availability error. The request exceeds the switch's limit for outstanding requests.
ctcOverallMonLimEx	SWITCH	System resource error. The request exceeded the switch's overall limit for monitors.
ctcPacErr	SWITCH	Security error. The switch has detected an error in the privilege attribute certificate.

Table 3–1 Condition Values Returned (Continued)

Error	From	Description
ctcParseError	CTC SERVER	The CTC server could not parse the message from the switch. This indicates an internal error. Report the problem to Dialogic.
ctcPerfGeneric	SWITCH	General performance management error. The CTC server is unable to provide more specific information.
ctcPerfLimEx	SWITCH	Performance management error. A performance limit has been exceeded.
ctcPrivateCstaErr	SWITCH	Unspecified error. The CSTA switch has returned a non-standard error. Report the problem to Dialogic and your switch manufacturer.
ctcPrivViolCalledDev	SWITCH	Operation error. The request cannot be provided because the called device is not authorized for the service.
ctcPrivViolCallingDev	SWITCH	Operation error. The request cannot be provided because the calling device is not authorized for the service.
ctcPrivViolSpecDev	SWITCH	Operation error. The request cannot be provided because the specified device is not authorized for the service.
ctcRcvReqRej	CTC SERVER	The switch rejected a message or request from the CTC server. This indicates an internal error. Report the problem to Dialogic.
ctcReadError	CTC SERVER	The read data request on the link between the CTC server and the switch has returned an error. This could indicate that the switch has stopped or restarted the link, or that there may be a problem with the link hardware.
ctcReqIncomWithCalledDev	SWITCH	The requested CSTA service is not compatible with the called device. This error is returned by CSTA Phase II switches only.
ctcReqIncomWithCallingDev	SWITCH	The requested CSTA service is not compatible with the calling device. This error is returned by CSTA Phase II switches only.
ctcReqIncomWithObj	SWITCH	Operation error. The request is incompatible with the object.

Table 3–1 Condition Values Returned (Continued)

Error	From	Description
ctcResAllocError	SERVER	Resource allocation error.
ctcResourceBusy	SWITCH	System resource error. An internal resource is temporarily busy.
ctcResOutOfServ	SWITCH	System resource error. The service requires a resource that is out of service. This could indicate a resource problem; check with your network administrator.
ctcRouteDataLost	CTC SERVER	A number of calls have been presented to the application at the same time and some route data has been lost.
ctcRouteInProgress	CTC API	A previous ctcGetRouteQuery or ctcWinGetRouteQuery request has not yet completed.
ctcRoutingOff	CTC API	The call to ctcGetRouteQuery or ctcWinGetRouteQuery has been returned because the specified channel is deassigned.
ctcRPCConnecFail	CTC API	An RPC error was received. Due to the error, an RPC network connection cannot be created and the CTC client cannot communicate with the CTC server.
ctcSealErr	SWITCH	Security error. The switch has detected an error in the seal.
ctcSecGeneric	SWITCH	General security error. The switch is unable to provide more specific information.
ctcSecurityViol	SWITCH	Operation error. The request violates a security requirement.
ctcSeqNumErr	SWITCH	Operation error. The switch has detected an error in the sequence number of the operation.
ctcServerUnknown	CTC API	The specified system is not a CTC server. Check that the specified name is correct and that the CTC Server software is installed and running on the system.
ctcServerBusy	CTC API	The CTC server is too busy to respond, possibly because it is shutting down.
ctcServiceBusy	SWITCH	System resource error. The requested service is supported by the switch but is temporarily unavailable.

Table 3–1 Condition Values Returned (Continued)

Error	From	Description
ctcStGeneric	SWITCH	State incompatibility error. The service request was not compatible with the condition of a related CSTA object. The switch is unable to provide more specific information.
ctcSubsGeneric	SWITCH	Generic subscribed resource availability error. The switch is unable to provide more specific information.
ctcSuccess	CTC API	The routine completed successfully.
ctcSysGeneric	SWITCH	General system resource availability error. The switch is unable to provide more specific information.
ctcSwitchDisabled	SWITCH	The switch is disabled.
ctcSwitchEnabled	SWITCH	The switch is enabled.
ctcSwitchInit	SWITCH	The switch is initializing.
ctcSwitchOverImm	SWITCH	Switch overload is imminent.
ctcSwitchOverRch	SWITCH	Switch overload has been reached.
ctcSwitchOverRel	SWITCH	Switch overload is relieved.
ctcTimeout	CTC SERVER	The switch did not respond to the request from the CTC server. There may be a problem with the link between the CTC server and the switch or the switch may be too busy to respond. If there is a problem with the link, this value may be returned each time the CTC server checks the state of the link. The Retry Count set with the Configuration Program specifies the number of times the CTC server checks the link. For more information, refer to the *CT-Connect Installation and Administration Guide* for your CTC server platform.
ctcTimeStampErr	SWITCH	Security error. The switch has detected an error in the time stamp of the operation.
ctcUCBFail	CTC SERVER	The UCB initialization procedure detected insufficient virtual memory on the CTC server.
ctcUnimplemented	CTC API	The option is not implemented on the switch.

Table 3–1 Condition Values Returned (Continued)

Error	From	Description
ctcUnspecCstaErr	SWITCH	The switch has detected an error of an unspecified type. This value is returned by CSTA switches only.
ctcUnspecified	SWITCH	The switch has detected an unspecified error.
ctcUnsupAPIversion	CTC SERVER	Either the version of CTC Server software running on the CTC server is not compatible with the version of CTC API running on your CTC client, or you did not pass a valid value in the APIversion field of the ctcAssignData structure. For more information, refer to the description of ctcAssign in Chapter 2.
ctcUnsupProc	CTC SERVER	The specified procedure is not supported for the assigned device.
ctcValueOutOfRange	SWITCH	Operation error. The parameter contains a value that is not in the range defined for the switch.
ctcXmitError	CTC SERVER	The send data request on the link between the CTC server and the switch has returned an error condition.

Part II

Part II consists of four appendixes:

- Appendix A lists the CTC features that are common to all supported switch protocols and switches.
- Appendix B lists CTC API features specific to switches that support the CSTA protocol. It also describes CSTA-specific CTC routines for applications that work with CSTA switches only.
- Appendix C describes features of the CTC API that are specific to the link with Lucent DEFINITY Generic 3 (G3) switches. It also describes the DEFINITY-specific CTC routine for applications that work with the Lucent DEFINITY G3 only.
- Appendix D describes features of the CTC API that are specific to the link with Nortel Meridian switches. It also describes the Meridian-specific routines for CTC applications that work with a Meridian switch only.

A

Features Common to All CTC Protocol/Switch Links

Dialogic's CTC API is designed to provide a compatible interface for all of the supported protocols/switches. However, because there is no standard set of features offered by the switch manufacturers, not all of the functions documented in Chapter 2 will be available with each type of supported protocol/switch.

This appendix provides a summary of:

- The level of support for each CTC function by protocol/switch (see Table A–1)
- The CTC functions that are compatible with all of the supported protocol/switches (see Section A.1)

Read this appendix in conjunction with the appendixes that follow, each of which indicate the functions that are specific to an individual protocol or switch. Comparing this appendix and protocol/switch-specific appendixes should indicate the amount of work involved in modifying your application if you want to use all of the features available at this release (and then, in the future, want the application to work with another protocol or switch).

The information provided on CSTA represents Dialogic's implementation of the CSTA protocol developed by the ECMA standards group. If you are using this protocol with CTC, also consult the documentation provided with your switch to determine which features are supported by your switch.

Table A–1 indicates the level of support provided by the individual protocols and links supported at this version of CTC.

Table A–1 Protocol/Switch-Specific Support for CTC Routines

	CSTA I	CSTA II	DEFINITY	Meridian
ctcAddMonitor	Y	Y	Y	Y
ctcAnswerCall	Y	Y	Y	Y†
ctcAssign	*	*	*	*
ctcAssociateData	*	Y	N	N
ctcCancelCall	Y	Y	*	N‡
ctcConferenceJoin	Y	Y	Y	Y
ctcConsultationCall	*	*	Y	*
ctcDeassign	Y	Y	Y	Y
ctcDeflectCall	*	*	*	N
ctcErrMsg	Y	Y	Y	Y
ctcGetAgentStatus	Y	Y	*	N
ctcGetCallForward	*	*	*	N
ctcGetChannelInformation	*	*	*	*
ctcGetDoNotDisturb	Y	Y	Y	N
ctcGetEvent	*	*	*	*
ctcGetMessageWaiting	Y	Y	Y	N
ctcGetMonitor	Y	Y	Y	Y
ctcGetRouteQuery	*	*	*	*
ctcGetRoutingEnable	N	Y	N	N
ctcHangupCall	Y	Y	Y	Y
ctcHoldCall	Y	Y	Y	*
ctcMakeCall	*	*	*	*

†Not supported for channels assigned to 500 or 2500 sets.
‡Refer to the switch-specific appendix for more information.

Y–Supported as documented in Chapter 2.
N–Not Supported.
*–Supported as noted in the switch-specific appendixes.

Table A–1 Protocol/Switch-Specific Support for CTC Routines (Continued)

	CSTA I	CSTA II	DEFINITY	Meridian
ctcMakePredictiveCall	*	*	*	N
ctcPickupCall	Y	Y	N	N
ctcReconnectHeld	Y	Y	Y	Y
ctcRemoveMonitor	Y	Y	Y	Y
ctcRespondToInactiveCall	Y	Y	N	N
ctcRespondToRouteQuery	*	*	*	*
ctcRetrieveHeld	Y	Y	Y	*
ctcSendDTMF	N	Y	*	N
ctcSetAgentStatus	*	*	*	*
ctcSetCallForward	*	*	*	*
ctcSetDoNotDisturb	Y	Y	*	Y
ctcSetMessageWaiting	Y	Y	Y	Y
ctcSetMonitor	Y	Y	Y	Y
ctcSetRoutingEnable	N	Y	N	N
ctcSingleStepTransfer	N	Y	N	*
ctcSnapshot	Y	Y	*	N
ctcSwapWithHeld	Y	Y	Y	N
ctcTransferCall	Y	Y	Y	*
ctcWinGetEvent	*	*	*	*
ctcWinGetRouteQuery	*	*	*	*

Y–Supported as documented in Chapter 2.
N–Not Supported.
*–Supported as noted in the switch-specific appendixes.

A.1 Common CTC Functions

The following routines are supported by all the protocol/switches CTC currently supports:

ctcAddMonitor
ctcAnswerCall
ctcAssign
ctcConferenceJoin
ctcConsultationCall
ctcDeassign
ctcErrMsg
ctcGetChannelInformation
ctcGetEvent
ctcGetMonitor
ctcGetRouteQuery
ctcHangupCall
ctcHoldCall
ctcMakeCall
ctcReconnectHeld
ctcRemoveMonitor
ctcRespondToRouteQuery
ctcRetrieveHeld
ctcSetAgentStatus
ctcSetCallForward
ctcSetDoNotDisturb
ctcSetMessageWaiting
ctcSetMonitor
ctcTransferCall
ctcWinGetEvent
ctcWinGetRouteQuery

Generally, if you write an application using these functions, you will probably need to make only a few changes for it to work with more than one of the supported protocol/switches.

However, there may be some aspects of these routines that may not be common to all switches, for example, monitoring (see Section A.2). For more information, refer to the following switch-specific appendixes.

Also note that a switch using the CSTA protocol may not support all of these functions. For example, not all CSTA switches support the ctcSetCallForward routine which lets you set conditions for forwarding calls for the assigned device. Refer to the documentation provided with your CSTA switch for details of the features that it supports.

A.2 Monitoring

The monitoring information returned by ctcGetEvent and ctcWinGetEvent can include information on call states, call events, event qualifiers, call types, call parties and party qualifiers:

- Call States—the call states documented in Chapter 2 for ctcGetEvent and ctcWinGetEvent should be common to all links. Some switches may return more information on call states; refer to the appropriate appendix for more information.
- Call Events and Qualifiers—all switches return some information on call events, but the level of information returned will vary depending on the switch. The fields in the ctcEventData structure are common to all switches. However, some switches provide more information than others on call events, and this extra, switch-specific, information is returned in the eventQualifier field.
- Call Types—not all switches support call types. Refer to your switch-specific appendix for information about call events and call types.
- Call Parties—most switches return some information on the other party, third party, and called party involved in an event. Some switches also provide a qualifier, providing additional identifying information on these parties. Refer to the appropriate switch-specific appendix for details about party information and qualifiers.

B

Features Specific to the CSTA Protocol

This appendix describes Dialogic's implementation of the Computer Supported Telephony Applications (CSTA) protocol, Phase I and Phase II, as developed by the ECMA standards group. The appendix identifies those CTC routines that are supported by CSTA Phase I and Phase II, and notes which routines work differently from the descriptions in Chapter 2.

Use this appendix in conjunction with the documentation provided with your switch to determine which features are supported by your switch.

B.1 Standard CTC Functions Supported by CSTA

Table B–1 lists the standard CTC routines supported by CSTA. If a routine is listed as *Supported as noted*, its limitations are documented in this appendix. CSTA. If a routine is listed as *Supported fully*, it is supported as described in Chapter 2.

For details of the CSTA-specific routines available with CTC, refer to Section B.15.

Table B–1 CTC Functions Specific to CSTA

CTC Routine	Support
ctcAddMonitor	Supported fully.
ctcAnswerCall	Supported fully.
ctcAssign	Supported as noted in Section B.2.
ctcAssociateData	Supported as noted in Section B.3.
ctcCancelCall	Supported fully.
ctcConferenceJoin	Supported fully.
ctcConsultationCall	Supported as noted in Section B.4.
ctcDeassign	Supported fully.
ctcDeflectCall	Supported as noted in Section B.5.
ctcErrMsg	Supported fully.
ctcGetAgentStatus	Supported fully.
ctcGetCallForward	Supported as noted inSection B.6.
ctcGetChannelInformation	Supported as noted in Section B.7.
ctcGetDoNotDisturb	Supported fully.
ctcGetEvent	Supported as noted in Section B.8.
ctcGetMessageWaiting	Supported fully.
ctcGetMonitor	Supported fully.
ctcGetRouteQuery	Supported as noted in Section B.9.
ctcGetRoutingEnable	CSTA Phase I switches: not supported. CSTA Phase II switches: supported fully.
ctcHangupCall	Supported fully.

Table B–1 CTC Functions Specific to CSTA (Continued)

CTC Routine	Support
ctcHoldCall	Supported fully.
ctcMakeCall	Supported as noted in Section B.10.
ctcMakePredictiveCall	Supported as noted in Section B.11.
ctcPickupCall	Supported fully.
ctcReconnectHeld	Supported fully.
ctcRemoveMonitor	Supported fully.
ctcRespondToInactiveCall	Supported fully.
ctcRespondToRouteQuery	Supported as noted in Section B.12.
ctcRetrieveHeld	Supported fully.
ctcSendDTMF	CSTA Phase I switches: not supported. CSTA Phase II switches: supported fully.
ctcSetAgentStatus	Supported as noted in Section B.13.
ctcSetCallForward	Supported as noted in Section B.14.
ctcSetDoNotDisturb	Supported fully.
ctcSetMessageWaiting	Supported fully.
ctcSetMonitor	Supported fully.
ctcSetRoutingEnable	CSTA Phase I switches: not supported. CSTA Phase II switches: supported fully.
ctcSingleStepTransfer	CSTA Phase I switches: not supported. CSTA Phase II switches: supported fully.
ctcSnapshot	Supported fully.
ctcSwapWithHeld	Supported fully.
ctcTransferCall	Supported fully.
ctcWinGetEvent	Supported as noted in Section B.8.
ctcWinGetRouteQuery	Supported as noted in Section B.9.

Sections B.2 to B.16 point out the technical distinctions to note when writing applications that call the routines listed as *Supported as noted* in Table B–1. If you write an application that uses these features, or any of the CSTA-specific features described in Section B.15, you will have to modify it to work with other CTC-compatible switches.

B.2 ctcAssign

This section describes operating differences and points to note when you use ctcAssign with a CSTA switch. For a full description of this routine, refer to Chapter 2.

B.2.1 Supported Devices

You can assign a channel to the following devices:

- Voice sets (telephones)
- Groups (ACD groups, queues)
- Route points
- Monitor channels

B.2.2 Extension to the CTC API

If you are writing an application that works with CSTA switches only, you can use additional CSTA routines described in Section B.15. To make use of these routines, specify the following value in the APIextensions field of the ctcAssignData structure:

ctcK_CstaPrivate

B.2.3 Devices and Supported Routines

Table B–2 shows the routines supported for each type of device. A cross (X) indicates that the routine is supported.

Table B–2 Routines Supported for CSTA Switches

CTC Routine	Voice Set	Group	Route Point	Monitor Channel
ctcAddMonitor				X
ctcAnswerCall	X			
ctcAssign	X	X	X	X
ctcAssociateData	X			

Table B–2 Routines Supported for CSTA Switches (Continued)

CTC Routine	Voice Set	Group	Route Point	Monitor Channel
ctcCancelCall	X			
ctcConferenceJoin	X			
ctcConsultationCall	X			
ctcDeassign	X	X	X	X
ctcDeflectCall	X			
ctcErrMsg	X	X	X	X
ctcGetAgentStatus	X			
ctcGetCallForward	X			
ctcGetChannelInformation	X	X	X	X
ctcGetDoNotDisturb	X			
ctcGetEvent	X	X	X	X
ctcGetMessageWaiting	X			
ctcGetMonitor	X	X	X	
ctcGetRouteQuery			X	
ctcGetRoutingEnable			X	
ctcHangupCall	X			
ctcHoldCall	X			
ctcMakeCall	X	X		
ctcMakePredictiveCall		X		
ctcPickupCall	X			
ctcReconnectHeld	X			
ctcRemoveMonitor				X
ctcRespondToInactive	X			
ctcRespondToRouteQuery			X	
ctcRetrieveHeld	X			
ctcSendDTMF	X	X		
ctcSetAgentStatus	X			

Table B–2 Routines Supported for CSTA Switches (Continued)

CTC Routine	Voice Set	Group	Route Point	Monitor Channel
ctcSetCallForward	X			
ctcSetDoNotDisturb	X			
ctcSetMessageWaiting	X			
ctcSetMonitor	X	X	X	
ctcSetRoutingEnable			X	
ctcSingleStepTransfer	X	X		
ctcSnapshot	X	X		
ctcSwapWithHeld	X			
ctcTransferCall	X			
ctcWinGetEvent	X	X	X	
ctcWinGetRouteQuery			X	
CSTA-Specific Routines				
ctcCstaEscape	X	X		
ctcCstaGetPrivateData	X	X	X	
ctcCstaGetPrivateEventData	X	X		
ctcCstaGetPrivateRouteData			X	
ctcCstaSetPrivateData	X	X	X	

B.2.4 Assigning to ODNs and ADNs on Ericsson MD110 Digital Telephone Sets

If you are using ApplicationLink® 3.0 with the Ericsson MD110, you can use CTC to monitor Own Directory Numbers (ODNs) and Additional Directory Numbers (ADNs) on a Digital Telephone Set (DTS).

Monitoring ADNs and ODNs is dependent on the way that the MD110 ApplicationLink software is configured:

- If the MD110 ApplicationLink software is configured so that ADN/ODN monitoring is disabled (unchecked), you can monitor either an ODN or an ADN on the same DTS but not both.

  To assign to the ODN or ADN, pass ctcK_Dn in the deviceType field of the ctcAssignData structure and the ODN or ADN in the deviceDn field of the ctcAssignData structure.

- If the MD110 ApplicationLink software is configured so that ADN/ODN monitoring is enabled (checked), you can use CTC to monitor:
  - ADN

    If you pass only the ADN in the deviceDn field, your application will not be able to complete the following requests: ctcConferenceCall, ctcTransferCall, or ctcReconnectHeld. To monitor an ADN and complete these functions, you must assign to both the ADN and ODN on the DTS.

    Pass the following in the ctcAssignData structure:

    * ctcK_Dn in the deviceType field.
    * The ADN and ODN in the deviceDn field, using a colon (:) to separate the two numbers. For example, for ADN 2000 and ODN 3050 on the same DTS, specify 2000:3050 in the deviceDn field.
  - ODN

    To monitor the ODN only, pass ctcK_Dn in the deviceType field, and the ODN in the deviceDn field, of the ctcAssignData structure.

For more information about configuring the MD110 ApplicationLink software, refer to the *MD110 ApplicationLink 3.0 Application Programmer's Guide*.

B.3 ctcAssociateData

This routine is supported for:

- Switches supporting CSTA Phase II, for example, the Alcatel 4400
- Alcatel switches supporting CSTA Phase I
- Ericsson MD110 (BC9) switches supporting CSTA Phase I

The Siemens Hicom 300E does not support ctcAssociateData. However, you can use the applicationData argument of the ctcConsultationCall and ctcDeflectCall routines to associate data with a call. For more information, refer to the description of these routines in Chapter 2 and CSTA-specific differences in this appendix.

B.4 ctcConsultationCall

This section describes operating differences when you use ctcConsultationCall with a CSTA switch. For a full description of this routine, refer to Chapter 2.

B.4.1 Application Data

Support for application data is dependent on your switch:

For this type of switch...	Application Data is...
Siemens Hicom 300E supporting CSTA Phase I	Supported fully. For details of the applicationData argument, refer to the description of ctcConsultationCall in Chapter 2.
Other CSTA Phase I switch	Not supported. Pass the address of a zero-length character string with the applicationData argument.
CSTA Phase II switch	Supported fully. For details of the applicationData argument, refer to the description of ctcConsultationCall in Chapter 2.

B.5 ctcDeflectCall

This section describes operating differences when you use ctcDeflectCall with a CSTA switch. For a full description of this routine, refer to Chapter 2.

B.5.1 Application Data

Support for application data is dependent on your switch:

For this type of switch...	Application Data is...
Siemens Hicom 300E supporting CSTA Phase I	Supported fully. For details of the applicationData argument, refer to the description of ctcDeflectCall in Chapter 2.
Other CSTA Phase I switch	Not supported. Pass the address of a zero-length character string with the applicationData argument.
CSTA Phase II switch	Supported fully. For details of the applicationData argument, refer to the description of ctcDeflectCall in Chapter 2.

B.6 ctcGetCallForward

This section describes operating differences when you use ctcGetCallForward with a CSTA switch. For a full description of this routine, refer to Chapter 2.

B.6.1 Call-Forward Settings Returned

CTC returns the value ctcK_NoAnswerBusy with the forwardMode argument when the following call-forward settings are set on or off:

ctcK_CfExtBusy
ctcK_CfExtNoAnswer
ctcK_CfIntBusy
ctcK_CfIntNoAnswer
ctcK_CfNoAnswerBusy

B.7 ctcGetChannelInformation

This section describes operating differences and points to note when you use ctcGetChannelInformation with a CSTA switch. For a full description of this routine, refer to Chapter 2.

B.7.1 Line Types

The following values can be returned in the lineType field of the ctcChanData structure:

ctcK_LineACD
ctcK_LineDataSet
ctcK_LineMonitorChannel
ctcK_LineTrunk
ctcK_LineUnknown
ctcK_LineVoiceSet

The value ctcK_LineRoutePoint is not returned.

B.7.2 Set Types

The following values can be returned in the setType field of the ctcChanData structure:

ctcK_SetACD
ctcK_SetACDGroup
ctcK_SetButton
ctcK_SetButtonGroup
ctcK_SetLine
ctcK_SetLineGroup
ctcK_SetOperator
ctcK_SetOperatorGroup
ctcK_SetOther
ctcK_SetSetStation
ctcK_SetStationGroup
ctcK_SetTrunk

ctcK_SetTrunkGroup
ctcK_SetUnknown

B.7.3 Switch-Specific Support

If you are using CSTA-specific routines (see Section B.15), the following values can be returned in the switchSpecificSupport field of the ctcChanData structure:

ctcM_CstaEscape
ctcM_CstaGetPrivateData
ctcM_CstaGetPrivateEventData
ctcM_CstaGetPrivateRouteData
ctcM_CstaSetPrivateData

B.8 ctcGetEvent and ctcWinGetEvent

This section describes operating differences and points to note when you use ctcGetEvent or ctcWinGetEvent with a CSTA switch. For full descriptions of these routines, refer to Chapter 2.

B.8.1 Fields Used in the ctcEventData Structure

Table B–3 shows how the fields in the ctcEventData structure are supported for CSTA switches. If Table B–3 specifies that the field is *Not supported*, CTC always returns null data for that field.

Table B–3 Event Information Supported by CSTA Switches

Field	Support
refId	See Chapter 2.
netCallId	Not supported.
oldRefId	See Chapter 2.
oldNetCallId	Not supported.
state	See Chapter 2.
event	See Chapter 2, also Sections B.8.4 and B.8.7.
eventQualifier	See Section B.8.5.
type	Not supported.
otherPartyType	See Chapter 2.
otherPartyQualifier	See Section B.8.6.
otherParty	See Section B.8.7.

Table B–3 Event Information Supported by CSTA Switches (Continued)

Field	Support
otherPartyTrunk	See Chapter 2.
otherPartyGroup	Not supported.
thirdPartyType	See Chapter 2.
thirdPartyQualifier	See Section B.8.6.
thirdParty	See Section B.8.7.
thirdPartyTrunk	See Chapter 2.
thirdPartyGroup	Not supported.
calledPartyType	See Chapter 2.
calledPartyQualifier	See Section B.8.6.
calledParty	See Section B.8.7.
calledPartyTrunk	See Chapter 2.
calledPartyGroup	Not supported.
applicationData	Supported by the following switches only: • CSTA Phase II switches • Alcatel switches (CSTA Phase I and Phase II) • Ericsson MD110 (BC9) • Siemens Hicom 300E For these switches, see Chapter 2.
monitorParty	See Chapter 2.
nestedMonitorChannel	See Chapter 2.
agentMode	See Section B.8.4.
agentId	See Section B.8.4.
agentGroup	See Section B.8.4.
agentData	See Section B.8.4.
logicalAgent	Not supported.
dtmfDigits	Supported by CSTA Phase II switches only. For these switches, see Chapter 2.
originatingPartyType	Supported by CSTA Phase II switches only. For these switches, see Chapter 2.

Table B–3 Event Information Supported by CSTA Switches (Continued)

Field	Support
originatingPartyQualifier	Not supported.
originatingParty	Supported by CSTA Phase II switches only. For these switches, see Chapter 2.
originatingPartyTrunk	Supported by CSTA Phase II switches only. For these switches, see Chapter 2.
originatingPartyGroup	Not supported.
secOldRefId	See Chapter 2.
callsQueued	See Chapter 2.
accountInfo	Supported by CSTA Phase II switches only. For these switches, see Chapter 2.
timeStamp	See Section B.8.8.
privateData	Only supported by CSTA switches using the CSTA API extension. See the description of ctcCstaPrivateEventData in Section B.15.

B.8.2 Group Monitoring

If you are monitoring a channel assigned to a group (or queue), refer to your CSTA switch documentation for information about the states and events returned.

B.8.3 Return Values for Transient States

If the switch is in a transient state, it reports this state to CTC. This can cause CTC to return one of the following condition values for ctcGetEvent or ctcWinGetEvent:

ctcEventDataLost
ctcSwitchDisabled
ctcSwitchEnabled
ctcSwitchInit
ctcSwitchOverImm
ctcSwitchOverRch
ctcSwitchOverRel

If these values are returned, repost ctcGetEvent or ctcWinGetEvent.

B.8.4 Agent Events

Table B–4 shows agent information returned by CSTA Phase I switches for agent events. Table B–5 shows agent information returned by CSTA Phase II switches for agent events.

Table B–4 Agent Event Information Returned by CSTA Phase I Switches

Event	Field	Contents
ctcK_AgentModeChange		
	agentMode	One of the following values: ctcK_AgentReady ctcK_AgentNotReady ctcK_AgentOtherWork ctcK_AgentAfterCallWork
	agentId	Identifier for the agent
	agentGroup	Null
	agentData	Null
	logicalAgent	Null
ctcK_AgentLoggedOn		
	agentMode	Null
	agentId	Identifier for the agent
	agentGroup	DN for the group
	agentData	Agent information (for example, a password)
	logicalAgent	Null
ctcK_AgentLoggedOff		
	agentMode	Null
	agentId	Identifier for the agent
	agentGroup	DN for the group
	agentData	Null
	logicalAgent	Null

Table B–5 Agent Event Information Returned by CSTA Phase II Switches

Event	Field	Contents
ctcK_AgentModeChange		
	agentMode	One of the following values: ctcK_AgentBusy ctcK_AgentReady ctcK_AgentNotReady ctcK_AgentAfterCallWork
	agentId	Identifier for the agent
	agentGroup	For ctcK_AgentBusy and ctcK_AgentAfterCallWork modes, this field returns the DN for the group. For ctcK_AgentReady and ctcK_AgentNotReady modes, this field returns null data.
	agentData	Null
	logicalAgent	Null
ctcK_AgentLoggedOn		
	agentMode	Null
	agentId	Identifier for the agent
	agentGroup	DN for the group
	agentData	Agent information (for example, a password)
	logicalAgent	Null
ctcK_AgentLoggedOff		
	agentMode	Null
	agentId	Identifier for the agent
	agentGroup	DN for the group
	agentData	Agent information (for example, a password)
	logicalAgent	Null

B.8.5 Call Event Qualifiers for CSTA

This section describes the CSTA qualifiers for call events. Call events occur during the progress of a call and, along with call states, indicate the success or failure of calls involving the monitored device. The qualifier provides more information on the nature of the event.

CTC returns additional information about call events in the eventQualifier field of the ctcEventData structure.

To determine which event has occurred, compare the value returned in the eventQualifier field with the literals listed as ctcK_Eq... in Table B–6.

The literals define the possible qualifiers returned by CSTA and are supplied in the definitions file installed on your system.

Table B–6 Call Event Qualifiers for CSTA

Qualifier Literal	Description
ctcK_EqActiveMonitor	An Active Monitor feature (Unannounced Barge In) has occurred. This feature typically allows intrusion by a supervisor into an agent call with the ability to speak and listen, and without any tone to signal the intrusion. Because the resultant call can be considered a conference call, this qualifier can be supplied with a conference event.
ctcK_EqAlternate	The call is in the process of being exchanged. This feature is typically found on single-line telephones where the user is required to press the switch hook to put one call on hold and retrieve a held call, or to answer a waiting call.
ctcK_EqBlocked	One party has disconnected from a call leaving one other party remaining with a local connection. This qualifier is returned for CSTA Phase II switches only.
ctcK_EqBusy	The call encountered a busy tone or device.
ctcK_EqCallBack	The Call Back feature was invoked, to complete a call that has encountered a busy or no answer condition. As a result, the failed call is cleared and the call considered as queued, and the switch will subsequently retry the call. Consequently, this qualifier can be supplied for events relating to any stage of the call process.
ctcK_EqCallCancelled	The user has terminated a call without going on-hook.

Table B–6 Call Event Qualifiers for CSTA (Continued)

Qualifier Literal	Description
ctcK_EqCallForwardAlways	The call has been redirected by setting Call Forwarding for all conditions.
ctcK_EqCallForwardBusy	The call has been redirected by setting Call Forwarding for a busy endpoint.
ctcK_EqCallForwardNoAnswer	The call has been redirected by setting Call Forwarding for an endpoint that does not answer.
ctcK_EqCallForward	The call has been redirected by setting Call Forwarding for general, unknown, or multiple conditions.
ctcK_EqCallNotAnswered	The call was not answered because a timer has elapsed.
ctcK_EqCallPickup	The call has been redirected by means of Call Pickup.
ctcK_EqCampOn	A Camp On feature has been invoked or has matured.
ctcK_EqConsultation	A consultation call is in progress. This qualifier is returned for CSTA Phase II switches only.
ctcK_EqDestNotObtainable	The call could not obtain the destination.
ctcK_EqDistributed	The call was distributed by an ACD or hunt group. This qualifier is returned for CSTA Phase II switches only.
ctcK_EqDoNotDisturb	The call has encountered a Do-Not-Disturb condition.
ctcK_EqEnteringDistribution	The call was delivered to a distribution mechanism (ACD). This qualifier is returned for CSTA Phase II switches only.
ctcK_EqForcedPause	The agent paused work. Regulations may require agents to have a period of time between handling successive ACD calls. This event qualifier is supported by CSTA Phase II switches only. It it returned with the ctcK_AgentModeChange event when the agent's work mode changes to Agent Not Ready (agentMode ctcK_AgentNotReady).
ctcK_EqIncompatibleDest	The call encountered an incompatible destination.
ctcK_EqInvalidAccountCode	The call became associated with an invalid account code.

Table B–6 Call Event Qualifiers for CSTA (Continued)

Qualifier Literal	Description
ctcK_EqKeyOperation	The event occurred at a bridged or twin device, where a telephone number associated primarily with one device is also associated with a second device.
ctcK_EqLockout	The call encountered inter-digit timeout while dialing.
ctcK_EqMaintenance	The call encountered a facility or endpoint in a maintenance condition.
ctcK_EqMakeCall	The event was in response to a make call. This qualifier may indicate that the caller is being prompted. It is returned for CSTA Phase II switches only.
ctcK_EqNetworkCongestion	The call encountered a congested network or switch. This may indicate that the user is listening to a “No Circuit” Special Information Tone (SIT) from a network, accompanied by an “All circuits are busy...” message.
ctcK_EqNetworkNotObtainable	The call could not reach a destination network.
ctcK_EqNetworkSignal	A network signal (trunk supervision or call progress) occurred. This qualifier is supported for CSTA Phase II switches only.
ctcK_EqNewCall	The call has not yet been redirected.
ctcK_EqNoAvailableAgents	The call could not access any agent.
ctcK_EqNormalClearing	The call or connection cleared in a normal way. This qualifier is supported for CSTA Phase II switches only.
ctcK_EqNumberChanged	The called number has been changed to a new number. This qualifier is supported for CSTA Phase II switches only.
ctcK_EqOverflow	The call overflowed a group queue or target.
ctcK_EqOverride	The call resulted from an encounter with the Override feature.

Table B–6 Call Event Qualifiers for CSTA (Continued)

Qualifier Literal	Description
ctcK_EqPark	The Call Park feature has been set at the device, placing or retrieving a call in a parked position. Placing a call in a park position releases the call from the parking device, but retains the call in the switching function to be connected to another (or same) device by invoking the unparking feature there.
ctcK_EqRecall	The Recall feature has been set at the device. This feature alerts a device that a timeout has failed to complete, or that further user action is anticipated.
ctcK_EqRedirected	The call has been redirected.
ctcK_EqReorderTone	The call encountered a reorder tone, indicating that the request was not recognizable. This usually occurs when a user dials an invalid number or tries to obtain a service not enabled for that user or device. This qualifier can also indicate that the user is listening to a “Reorder” Special Information Tone (SIT), accompanied by a “The call did not go through as dialed...” message.
ctcK_EqResourcesNotAvail	Indicates resources were not available.
ctcK_EqSilentMonitor	A Silent Monitor feature has been set at the device. When a third party (such as an ACD agent supervisor) has joined a call, the feature ensures that the original party cannot hear the third party. A tone may be provided to one or both parties indicating that they are being monitored.
ctcK_EqSingleStepConference	A single-step conference occurred at the device. This qualifier is supported for CSTA Phase II switches only.
ctcK_EqSingleStepTransfer	The transfer was a single-step transfer. This qualifier is supported for CSTA Phase II switches only.
ctcK_EqTimeout	A trunk timer expired. This qualifier is supported for CSTA Phase II switches only.
ctcK_EqTransfer	A transfer is in progress or has occurred.
ctcK_EqTrunksBusy	The call encountered Trunks Busy.

Table B–6 Call Event Qualifiers for CSTA (Continued)

Qualifier Literal	Description
ctcK_EqVoiceUnitInitiator	The event resulted from action by automated equipment (voice mail device, voice response unit, announcement), rather than from direct action by a user.

B.8.6 Other, Third, and Called Party Qualifiers

Additional information about a party or called number can be returned in the otherPartyQualifier, thirdPartyQualifier, and calledPartyQualifier fields of the ctcEventData structure. These fields can contain one of the following:

ctcK_ConfController
ctcK_ReleasingDevice
ctcK_AlertingDevice
ctcK_CallingDevice
ctcK_CalledNumber
ctcK_LastRedirection
ctcK_NewRedirection
ctcK_AnsweringDevice
ctcK_HoldingDevice
ctcK_QueueNumber
ctcK_RetrievingDevice
ctcK_TransferringDevice
ctcK_DivertingDevice
ctcK_FailedDevice
ctcK_TrunkUsed
ctcK_AddedParty

B.8.7 Party Information for Call Events

Table B–7 provides details of the party information returned for call events.

Table B–7 CSTA Party Information for Call Events

Event	Field	Party Information	Explanation
ctcK_DestBusy			
	Other	Busy Party	
	Third	Null	
	Called	Called Number	
	Originating	Null	

Table B–7 CSTA Party Information for Call Events (Continued)

Event	Field	Party Information	Explanation
ctcK_DestChanged			
	Other	Ringing Party	
	Third	Previous Destination	The first station called.
	Called	Called Number	
	Originating	Null	
ctcK_DestInvalid			
	Other	Invalid Destination Number	
	Third	Null	
	Called	Called Number	
	Originating	Null	
ctcK_DestNotObtainable			
	Other	Null	
	Third	Null	
	Called	Called Number	
	Originating	Null	
ctcK_DestSeized			
	Other	Ringing Party	
	Third	Null	
	Called	Called Number	
	Originating	Point at which the call left the switch.	Returned by CSTA Phase II switches only.
ctcK_Diverted			
	Other	New Destination	
	Third	Null	
	Called	Null	
	Originating	Null	

Table B–7 CSTA Party Information for Call Events (Continued)

Event	Field	Party Information	Explanation
ctcK_Error			
	Other	Null	
	Third	Null	
	Called	Null	
	Originating	Null	
ctcK_InboundCall			
	Other	Calling Party	
	Third	Last Redirected	If the call was forwarded, the last station called before redirection.
	Called	Called Number	
	Originating	Point at which the call entered the switch.	Returned by CSTA Phase II switches only.
ctcK_Offhook			
	Other	Called Number	
	Third	Null	
	Called	Null	
	Originating	Null	
ctcK_OffhookPrompt			
	Other	Null	
	Third	Null	
	Called	Null	
	Originating	Null	

Table B–7 CSTA Party Information for Call Events (Continued)

Event	Field	Party Information	Explanation
ctcK_OpAnswered			
	Other	Answering Party	
	Third	Null	
	Called	ACD DN	When a call goes to a group queue.
	Originating	Point at which the call left the switch.	Returned by CSTA Phase II switches only.
ctcK_OpConferenced			
	Other	Conference Controller	The controller initiates a conference.
	Third	Null	
	Called	Null	
	Originating	Null	
ctcK_OpDisconnected			
	Other	Disconnecting Party	
	Third	Null	
	Called	Null	
	Originating	Null	
ctcK_OpHeld			
	Other	Holding Party	
	Third	Null	
	Called	Null	
	Originating	Null	
ctcK_OpRetrieved			
	Other	Retrieving Device	When a call on hold is picked up.
	Third	Null	
	Called	Null	
	Originating	Null	

Table B–7 CSTA Party Information for Call Events (Continued)

Event	Field	Party Information	Explanation
ctcK_Other			
	Other		Party information supplied depends on the event qualifier.
	Third		
	Called	Null	
	Originating	Null	
ctcK_TpAnswered			
	Other	Party at Other End of Call	
	Third	ACD DN or Ringing Number	
	Called	Called Number	
	Originating	Point at which the call entered the switch.	Returned by CSTA Phase II switches only.
ctcK_TpConferenced			
	Other	Null	
	Third	Null	
	Called	Null	
	Originating	Null	
ctcK_TpDisconnected			
	Other	Active Party Disconnected	
	Third	Any Held Party	Any party on hold.
	Called	Null	
	Originating	Null	

Table B–7 CSTA Party Information for Call Events (Continued)

Event	Field	Party Information	Explanation
ctcK_TpRetrieved			
	Other	New Active Party	When a call on hold is picked up.
	Third	Null	
	Called	Null	
	Originating	Null	
ctcK_TpSuspended			
	Other	Null	
	Third	Null	
	Called	Null	
	Originating	Null	
ctcK_Transferred			
	Other	New Party Connected	When the original party transfers a call in progress to a new party.
	Third	Party Making Transfer	
	Called	Null	
	Originating	Null	

This table gives only CSTA-specific information. For other switch-specific information, refer to the appendix for your switch (for example, Appendix C for the DEFINITY G3). For a description of each event (such as ctcK_DestBusy), see Table 2–5.

B.8.8 Timestamp

If the link is configured to return a timestamp from the CTC server, this field contains the date and time the CTC server processed the event.

If the link is configured to return a timestamp from the switch, this field contains:

- For CSTA Phase I switches, null data. Timestamp information is not supported by CSTA Phase I switches.
- For CSTA Phase II switches, the date and time the event was processed by

the switch.

For details of how to use the Configuration Program to change the configuration of a link, refer to the *CT-Connect Installation and Administration Guide* for your CTC server platform.

B.9 ctcGetRouteQuery and ctcWinGetRouteQuery

This section describes operating differences and points to note when you use ctcGetRouteQuery or ctcWinGetRouteQuery with a CSTA switch. For full descriptions of these routines, refer to Chapter 2.

B.9.1 Fields Used in the ctcRouteData Structure

Table B–8 shows how the fields in the ctcRouteData structure are supported for CSTA switches. If Table B–8 specifies that the field is *Not supported*, CTC always returns null data for that field.

Table B–8 Route Information Supported by CSTA Switches

Field	Support
routeId	See Chapter 2.
refId	See Chapter 2.
spare001	Not supported.
otherPartyType	See Chapter 2.
otherParty	See Chapter 2.
otherPartyTrunk	See Chapter 2.
otherPartyGroup	Not supported.
thirdPartyType	Not supported.
thirdParty	Not supported.
thirdPartyTrunk	Not supported.
thirdPartyGroup	Not supported.
calledPartyType	Not supported.
calledParty	Not supported.
calledPartyTrunk	Not supported.
calledPartyGroup	Not supported.

Table B–8 Route Information Supported by CSTA Switches (Continued)

Field	Support
applicationData	Supported by CSTA Phase II switches only. For these switches, see Chapter 2.
dtmfDigits	Not supported.
timeStamp	See Section B.9.2.
privateData	Only supported by CSTA switches using the CSTA API extension. See the description of ctcCstaPrivateRouteData in Section B.15.

B.9.2 Timestamp

If the link is configured to return a timestamp from the CTC server, this field contains the date and time the CTC server processed the route request.

If the link is configured to return a timestamp from the switch, this field contains:

- For CSTA Phase II switches, the date and time the request was processed by the switch.
- For CSTA Phase I switches, null data. Timestamp information is not supported by CSTA Phase I switches.

For details of how to use the Configuration Program to change the configuration of a link, refer to the *CT-Connect Installation and Administration Guide* for your CTC server platform.

B.10 ctcMakeCall

This section describes operating differences when you use ctcMakeCall with a CSTA switch. For a full description of this routine, refer to Chapter 2.

B.10.1 Application Data

Support for application data is dependent on your switch:

If your switch supports...	Then...
CSTA Phase I	Application data is not supported. Pass the address of a zero-length character string with the applicationData argument.

If your switch supports...	Then...
CSTA Phase II	Application data is supported fully. For details of the applicationData argument, refer to the description of ctcMakeCall in Chapter 2.

B.11 ctcMakePredictiveCall

This section describes operating differences and points to note when you use ctcMakePredictiveCall with a CSTA switch. For a full description of this routine, refer to Chapter 2.

B.11.1 Allocation Argument

Specify one of the following values with the allocation argument:

Value	Description
ctcK_AllocDefault	Enables the switch's default processing for the call.
ctcK_AllocActive	Specifies that the call is successful when the called device answers the call.
ctcK_AllocDelivered	Specifies that the call is successful when the call is put through to the device (for example, when the phone rings).

B.11.2 Application Data

Support for application data is dependent on your switch:

If your switch supports...	Then...
CSTA Phase I	Application data is not supported. Pass the address of a zero-length character string with the applicationData argument.
CSTA Phase II	Application data is supported fully. For details of the applicationData argument, refer to the description of ctcMakePredictiveCall in Chapter 2.

B.11.3 Number of Rings

The numberOfRings argument is not supported by CSTA switches. The value that you pass with the numberOfRings argument is not used by the switch.

B.12 ctcRespondToRouteQuery

This section describes operating differences when you use ctcRespondToRouteQuery with a CSTA switch. For a full description of this routine, refer to Chapter 2.

B.12.1 Application Data

Support for application data is dependent on your switch:

If your switch supports...	Then...
CSTA Phase I	Application data is not supported. Pass the address of a zero-length character string with the applicationData argument.
CSTA Phase II	Application data is supported fully. For details of the applicationData argument, refer to the description of ctcRespondToRouteQuery in Chapter 2.

B.13 ctcSetAgentStatus

This section describes operating differences and points to note when you use ctcSetAgentStatus with a CSTA switch. For a full description of this routine, refer to Chapter 2.

B.13.1 Logging In Agents

A CSTA password, agent ID, or group number may be required by your CSTA switch for logging in an agent. You can pass this data with the ctcSetAgentStatus arguments. Specify:

- ctcK_AgentLogin as the value for agentMode
- A CSTA password with the agentData argument
- A CSTA agent ID with the logicalAgent argument
- A CSTA group number with the agentGroup argument

B.13.2 Logging Out Agents

A CSTA password, agent ID, or group number may be required by your CSTA switch for logging out an agent. You can pass this data with the ctcSetAgentStatus arguments. Specify:

- ctcK_AgentLogout as the value for agentMode.
- A CSTA password with the agentData argument. Passwords for logging out

agents are supported by CSTA Phase II switches only.

- A CSTA agent ID with the logicalAgent argument.
- A CSTA group number with the agentGroup argument.

B.13.3 Agent Mode Not Supported

CSTA Phase II switches do not support the agentMode value ctcK_AgentOtherWork.

B.14 ctcSetCallForward

This section describes operating differences when you use ctcSetCallForward with a CSTA switch. For a full description of this routine, refer to Chapter 2.

B.14.1 Supported Call-Forwarding Settings

CSTA does not support the ctcK_CfNoAnswerBusy value for the forwardMode argument.

The ctcOptNotSup condition value is returned if you specify ctcK_CfNoAnswerBusy.

B.15 CTC Routines for CSTA Switches

This section describes CTC routines that are provided as an extension to the standard CTC API for CSTA Phase I and Phase II switches.

These routines enable you to exchange private data with your CSTA switch. Private data is information that a switch provides or requires for specific call features that cannot be passed using standard CTC routines. The type of data and its content is dependent on the switch, so, to use the routines effectively, Dialogic recommends that you work closely with the switch manufacturer.

B.15.1 Requirements

To make use of the routines:

- Your switch must support CSTA Phase I or CSTA Phase II
- When you use ctcAssign to assign a channel, you must specify the value ctcK_CstaPrivate in the APIextensions field of the ctcAssignData structure. Refer to the description of ctcAssign in Chapter 2 for more information.

B.15.2 Format of Private Data

Private data is passed or retrieved using the privateDataArray argument, which is required by each CSTA-specific private data routine.

The data sent or returned using the privateDataArray argument is dependent on the type of switch you are using and which private data routine you call:

- Table B–9 describes the values supported by all CSTA Phase I and Phase II switches, and values that are specific to the Alcatel 4400, Ericsson MD110, and Hicom 300E switches. Using these values, you can determine the type of private data you exchange with the switch.
- Table B–10 shows the type of data supported by each private data routine.

B.15.3 privateDataArray Argument

This section describes the privateDataArray argument required by all CSTA private data routines.

privateDataArray

type: **ctcPrivateDataArray**
access: **read and write**
mechanism: **by reference**

privateDataArray is the address of a fixed-format structure that contains the private data. The structure is formatted as follows:

```
      ctcPrivateDataArray {
              ctcPrivateData privData [3];
      };
```

ctcPrivateDataArray is an array of three ctcPrivateData structures of the format:

```
ctcPrivateData {
      ctcPrivDataType privDataType;
   union
      {   ctcPrivateDataRaw       raw;
          ctcPrivateDataRaw       rawByManufacturer;
          ctcAlcDateAndTimeString alcDateAndTime;
          unsigned int            alcServiceOptions;
          ctcAlcAcdWaitingTime    alcAcdWaitingTime;
          unsigned int            alcNetworkTimeSlot;
          ctcAlcOther             alcOther;
          ctcDeviceString         hcmSoftHeldDevice;
          unsigned short          hcmCause;
          boolean                 hcmCallingPartyIsAni;
          ctcHcmForwardElement    hcmForwardEventParams;
          ctcHcmAssociateData     hcmAssociateData;
          unsigned int            hcmTrunkNumber;
          unsigned short          hcmAutoAnswerMode;
          ctcHcmStationType       hcmStationType;
          unsigned int            hcmCallsQueued;
```

```
          ctcHcmSetForwardElement hcmStationFrwrdingParams;
          unsigned short          hcmSystemFrwrdingType;
          ctcHcmForwardlist       hcmForwardList;
          ctcHcmRouteTrigger      hcmRouteTrigger;
          unsigned int            hcmRejectCall;
          unsigned short          hcmRoutingEndCause;
          ctcHcmAgentState        hcmAgentState;
          ctcErcEnterDTMF         ercEnterDTMF;
          ctcErcMessageDiversion  ercMessageDiversion;
          ctcErcAccountCode       ercAccountCode;
          ctcErcAuthCode          ercAuthCode;
          ctcErcPressProgKey      ercPressProgKey;
          ctcErcCancelCallback    ercCancelCallback;
          ctcErcSetAssociateData  ercSetAssociateData;
          ctcErcFwdACDGroup       ercFwdACDGroup;
          ctcErcEvtAssociateData  ercEvtAssociateData;
          ctcErcConnectionId      ercFreeQueuePos;

      }privDataValue;
};
```

Each ctcPrivateData structure contains two fields:

- A privDataType field which identifies the type of data passed
- A privDataValue field which contains data of the type identified by privDataType

The following subsections describe the privDataType and privDataValue fields.

privDataType

This field identifies the type of data passed. Use:

- Table B–9 to find the switch you are using, the privDataType values that can be passed, and a brief description of the data type.
- Table B–10 to find the type of data the switch can return for the CSTA private data routine called (if the routine returns private data from the switch) or the type of data you can pass with a routine (if you use the routine to send private data to the switch).

  Table B–10 lists the data type in alphabetical order, and provides a guideline only. For more information about the type of data supported by your switch, contact your switch manufacturer.

Table B–9 Private Data Type Values

For this switch...	This value is used...	To pass this type of data...
CSTA Phase I or Phase II		
	ctcK_PrivNone	No data
	ctcK_PrivRawByManufacturer	Raw data and a manufacturer identifier
CSTA Phase II		
	ctcK_PrivRaw	Raw data
Alcatel 4400 supporting CSTA Phase II		
	ctcK_PrivAlcDateAndTime	Date and time
	ctcK_PrivAlcServiceOptions	Service options
	ctcK_PrivAlcAcdWaitingTime	ACD waiting time
	ctcK_PrivAlcNetworkTimeSlot	Network time slot
	ctcK_PrivAlcOther	Other Alcatel private data
Ericsson MD110 (BC9)		
	ctcK_PrivErcAccountCode	Account code
	ctcK_PrivErcAuthCode	Authorization code
	ctcK_PrivErcCancelCallback	Request to cancel callback on a device
	ctcK_PrivErcEnterDTMF	DTMF tones
	ctcK_PrivErcEvtAssociateData	Dialable string associated with an external caller
	ctcK_PrivErcFreeQueuePos	Request to release the queue position for a deflected call, without returning the call to the queue
	ctcK_PrivErcFwdACDGroup	Request to forward an ACD group to another destination
	ctcK_PrivErcKeepQueuePos	Request to hold the queue position for a call that has been deflected to another destination temporarily (for example, to an IVR)

Table B–9 Private Data Type Values (Continued)

For this switch...	This value is used...	To pass this type of data...
	ctcK_PrivErcMessageDiversion	Request to set or cancel diversion to operator
	ctcK_PrivErcPressProgKey	Request to press key number on an MD110 telephone
	ctcK_PrivErcSetAssociateData	Request to associate a dialable string with an external caller
Siemens Hicom 300E		
	ctcK_PrivHcmAgentState	Current work mode for an agent
	ctcK_PrivHcmAssociateData	Application data
	ctcK_PrivHcmAutoAnswerMode	Request to enable or disable the AutoAnswer feature at the originating device
	ctcK_PrivHcmCallingPartyIsAni	Value that identifies the calling party as ANI
	ctcK_PrivHcmCallsQueued	Number of calls queued
	ctcK_PrivHcmCause	Additional qualifying data for an event
	ctcK_PrivHcmForwardEventParams	Notification that the call forwarding setting has changed
	ctcK_PrivHcmForwardList	Additional information about the call forwarding setting
	ctcK_PrivHcmRouteTrigger	Request trigger of all configured Route Control Groups (RCGs)
	ctcK_PrivHcmRejectCall	Request to reject a route call request
	ctcK_PrivHcmRoutingEndCause	Reason for a call route to end
	ctcK_PrivHcmSoftHeldDevice	Number for the device placed on hold
	ctcK_PrivHcmStationFrwrdParams	Additional parameters for the call forwarding setting at a station
	ctcK_PrivHcmStationType	Type of station
	ctcK_PrivHcmSystemFrwrdingType	Call forwarding setting on the switch
	ctcK_PrivHcmTrunkNumber	Trunk number

Table B–10 Data Types Supported by Private Data Routines

privDataType Value	Escape	GetPrivateData	GetPrivate EventData	GetPrivate RouteData	SetPrivateData
ctcK_PrivAlcAcdWaitingTime			X	X	
ctcK_PrivAlcDateAndTime	X				X
ctcK_PrivAlcNetworkTimeSlot			X	X	
ctcK_PrivAlcOther	X	X	X	X	X
ctcK_PrivAlcServiceOptions	X				X
ctcK_PrivErcAccountCode	X				
ctcK_PrivErcAuthCode	X				
ctcK_PrivErcCancelCallback	X				
ctcK_PrivErcEnterDTMF	X				
ctcK_PrivErcEvtAssociateData			X		
ctcK_PrivErcFreeQueuePos	X				
ctcK_PrivErcFwdACDGroup	X				
ctcK_PrivErcKeepQueuePos					X
ctcK_PrivErcMessageDiversion	X				

Table B–10 Data Types Supported by Private Data Routines (Continued)

privDataType Value	Escape	GetPrivateData	GetPrivate EventData	GetPrivate RouteData	SetPrivateData
ctcK_PrivErcPressProgKey	X				
ctcK_PrivErcSetAssociateData	X				
ctcK_PrivHcmAgentState		X			
ctcK_PrivHcmAssociateData			X	X	X
ctcK_PrivHcmAutoAnswerMode					X
ctcK_PrivHcmCallingPartyIsAni			X	X	
ctcK_PrivHcmCallsQueued		X			
ctcK_PrivHcmCause			X	X	
ctcK_PrivHcmForwardEventParams			X	X	
ctcK_PrivHcmForwardList		X			
ctcK_PrivHcmRouteTrigger	X				
ctcK_PrivHcmRejectCall	X				
ctcK_PrivHcmRoutingEndCause				X	
ctcK_PrivHcmSoftHeldDevice			X		

Table B–10 Data Types Supported by Private Data Routines (Continued)

privDataType Value	Escape	GetPrivateData	GetPrivate EventData	GetPrivate RouteData	SetPrivateData
ctcK_PrivHcmStationFrwrdParams			X		
ctcK_PrivHcmStationType		X			
ctcK_PrivHcmSystemFrwrdingType			X		
ctcK_PrivHcmTrunkNumber			X		
ctcK_PrivNone	X	X	X	X	X
ctcK_PrivRawByManufacturer	X	X	X	X	X
ctcK_PrivRaw	X	X	X	X	X

privDataValue

This field contains data of the type defined by privDataType. It contains one of the following union elements:

- **raw**

  This element identifies the data as raw data and is supported for all switches compatible with CSTA Phase II. It contains the address of a fixed-format structure that is formatted as follows:

```
ctcPrivateDataRaw {
   ctcManufacturerString  manufacturer [ctcMaxManufacturerLen];
   unsigned int           dataLen;
   unsigned char          data [ctcMaxPrivateDataLen];
};
```

The following table describes the fields in the ctcPrivateDataRaw structure:

Field	Description
manufacturer	This field is not used when exchanging raw data with the switch. If you are passing data to the switch, specify a zero-length character string.
dataLen	This 32-bit field identifies the number of bytes of data in the data field.
data	This field contains an array of data. This data must be ASN.1 encoded.

- **rawByManufacturer**

  This element identifies the data as raw data and includes details of the switch manufacturer. It is supported for all switches compatible with CSTA Phase I and Phase II, and contains the address of the following fixed-format structure:

```
ctcPrivateDataRaw {
   ctcManufacturerString  manufacturer [ctcMaxManufacturerLen];
   unsigned int           dataLen;
   unsigned char          data [ctcMaxPrivateDataLen];
};
```

The following table describes the fields in the ctcPrivateDataRaw structure:

Field	Description
manufacturer	This character string contains an identifier for the switch manufacturer passing the data. For example, 1.3.12.1276.12. If you call ctcEscape or ctcSetPrivateData, make sure that you pass the string in the correct format. See your switch manufacturer for details.
dataLen	This 32-bit field identifies the number of bytes of data in the data field.
data	This field contains an array of data. This data must be ASN.1 encoded.

- **alcDateAndTime**

  This character string contains the date and time to be set on the switch. It is supported for Alcatel 4400 switches only.

- **alcServiceOptions**

  This 32-bit integer sets one or more service options on the switch. It is supported for Alcatel 4400 switches only, and contains one or more of the following bitmasks:

  ctcM_AlcCallProgressToneInhibition
  ctcM_AlcHoldToneInhibition
  ctcM_AlcPriorityTransfer

  For more information about these options, contact Alcatel.

- **alcAcdWaitingTime**

  This element contains the waiting time for calls queued on the switch. It is supported for Alcatel 4400 switches only and is the address of the following fixed-format structure:

```
ctcAlcAcdWaitingTime {
       unsigned int    waitingTime;
       unsigned int    saturation;
};
```

  The following table describes the fields in this structure:

Field	Description
waitingTime	This 32-bit field contains the waiting time, in seconds, for calls queued on the Alcatel switch.
saturation	A non-zero value in this 32-bit field indicates that the Alcatel queue is at its maximum call load.

  For more information about the data in these fields, contact Alcatel.

- **alcNetworkTimeSlot**

  This 32-bit integer contains the time slot on which the associated call arrived.

- **alcOther**

  This element contains other, undefined private data for an Alcatel 4400 switch. It is the address of the following fixed-format structure:

```
ctcAlcOther{
       unsigned char    identifier [2];
       unsigned int     dataLen;
       unsigned char    data [44];
};
```

The following table describes the fields in the ctcAlcOther structure:

Field	Description
identifier	This is an array of 2 bytes of data. The type of data is dependent on the type of information you want to exchange with the Alcatel 4400 switch. For more information, contact Alcatel.
dataLen	This 32-bit field specifies, in bytes, the length of data passed.
data	This is an array of 44 bytes of data. The type of data is dependent on the type of information you want to exchange with the Alcatel 4400 switch. For more information, contact Alcatel.

- **hcmSoftHeldDevice**

  This null-terminated character string contains the number for the device placed on hold. It is supported for Siemens Hicom 300E switches only. The maximum length for ctcDeviceString is specified by the literal ctcMaxDnLen defined in a CTC definitions file (see Section 1.5).

- **hcmCause**

  This unsigned 16-bit integer provides additional information for events. It is supported for Siemens Hicom 300E switches only, and can contain one of the following values:

  ctcK_HcmCauseCSTASoftHold
  ctcK_HcmCauseCSTAHardHold
  ctcK_HcmCauseCSTALineHold
  ctcK_HcmCauseBackgroundHold
  ctcK_HcmCauseExclusiveHold
  ctcK_HcmCauseCSTASingleStepTransfer

- **hcmCallingPartyIsAni**

  This unsigned character specifies whether the calling party is an ANI (Automatic Number Identification) number. It is supported for Siemens Hicom 300E switches only.

- **hcmForwardEventParams**

  This element contains details of the new call forwarding setting for a device, when the setting has changed. It is supported for Siemens Hicom 300E switches only, and is defined in the following fixed-format structure:

```
ctcHcmForwardElement{
       ctcHcmForwardingType  hcmFwdType;
       unsigned short        stationFwdType;
       unsigned short        systemFwdType;
       ctcDeviceString       forwardDN;
};
```

The fields in this structure are:

– **hcmFwdType**

  This field identifies which of the stationFwdType and systemFwdType fields contain data. It contains one of the following values:

  * ctcK_HcmFwdStation to indicate that data is passed in the stationFwdType field.

  * ctcK_HcmFwdSystem to indicate that the hcmForwardType field contains a systemFwdType value.

– **stationFwdType**

  This unsigned short can contain one of the following values:

  ctcK_HcmStnFwdTypeImmediateOn
  ctcK_HcmStnFwdTypeImmediateIntOn
  ctcK_HcmStnFwdTypeImmediateExtOn
  ctcK_HcmStnFwdTypeNoAnsOn
  ctcK_HcmStnFwdTypeBusyOn

– **systemFwdType**

  This unsigned short can contain one of the following values:

  ctcK_HcmSysFwdTypeImmediateOn
  ctcK_HcmSysFwdTypeImmediateIntOn
  ctcK_HcmSysFwdTypeImmediateExtOn
  ctcK_HcmSysFwdTypeNoAnswerOn
  ctcK_HcmSysFwdTypeNoAnswerIntOn
  ctcK_HcmSysFwdTypeNoAnswerExtOn
  ctcK_HcmSysFwdTypeBusyOn
  ctcK_HcmSysFwdTypeBusyIntOn
  ctcK_HcmSysFwdTypeBusyExtOn
  ctcK_HcmSysFwdTypeDoNotDisturbOn
  ctcK_HcmSysFwdTypeDoNotDisturbIntOn
  ctcK_HcmSysFwdTypeDoNotDisturbExtOn

– **forwardDN**

This character string contains the number for the device. The maximum length for ctcDeviceString is specified by the literal ctcMaxDnLen defined in a CTC definitions file (see Section 1.5).

- **hcmAssociateData**

This element contains data associated with a call, such as customer reference or customer account information. It is supported for the Siemens Hicom 300E only and is defined in the following fixed-format structure:

```
ctcHcmAssociateData{
       unsigned short  dataLen;
       unsigned char   data [32];
};
```

The fields in this structure are:

Field	Description
dataLen	This 16-bit field specifies, in bytes, the length of data passed.
data	This is an array of 32 bytes of data. The type of data is dependent on the type of information you want to exchange with the Siemens Hicom 300E switch. For more information, contact Siemens.

- **hcmTrunkNumber**

This element is a 32-bit integer that contains a trunk number.

- **hcmAutoAnswerMode**

This element is a 16-bit integer that specifies the setting for the AutoAnswer feature at the originating device. Use one of the following values:

Value	Description
ctcK_HcmAutoAnswerAttempt	AutoAnswer will be attempted.
ctcK_HcmAutoAnswerDisable	AutoAnswer is disabled.

- **hcmStationType**

This element identifies the station type. It is supported for the Siemens

Hicom 300E only and is defined in the following fixed-format structure:

```
ctcHcmStationType{
       ctcHcmStationCategory      hcmStationCategory;
          union {
                 ctcHcmAnalogTypes          analogStation;
                 ctcHcmDigitalStationType   digitalStation;
                 ctcHcmOtherEquipmentTypes  otherEquipment;
          } hcmStationCtgy;
};
```

The fields in this structure are:

– **hcmStationCategory**

This field identifies the type of data passed in the hcmStationCtgy union. It contains one of the following values:

Value	Description
ctcK_HcmAnalogStation	Identifies the category of station as analog.
ctcK_HcmDigitalStation	Identifies the category of station as digital.
ctcK_HcmOtherEquipment	Identifies the category of station as another type of equipment.

– **hcmStationCtgy**

This union contains one of the following elements:

* **analogStation**

This element identifies the type of analog station. It contains one of the following values:

ctcK_HcmAnlgGeneric
ctcK_HcmAnlgSet2500

* **digitalStation**

This element identifies the type of digital station and is defined as the following fixed-format structure:

```
ctcHcmDigitalStationType{
       ctcHcmDigitalTypes  digitalType;
       unsigned char       digitalAttributes;
};
```

The fields of the ctcHcmDigitalStationType structure are described in the following table:

Field	Description
digitalType	This field identifies the type of digital station. It contains one of the following values: ctcK_HcmDgtlGeneric ctcK_HcmDgtlSingleLine ctcK_HcmDgtlKeyset ctcK_HcmDgtlrp120 ctcK_HcmDgtlrp240 ctcK_HcmDgtlrp400 ctcK_HcmDgtlrp150 ctcK_HcmDgtlrp200 ctcK_HcmDgtlrp300 ctcK_HcmDgtlrp600 ctcK_HcmDgtlrp4327 ctcK_HcmDgtlset500 ctcK_HcmDgtlni1200 ctcK_HcmDgtloptie3 ctcK_HcmDgtloptie8 ctcK_HcmDgtloptieS ctcK_HcmDgtloptie1 ctcK_HcmDgtloptie1S ctcK_HcmDgtlset211 ctcK_HcmDgtlset260 ctcK_HcmDgtlset400 ctcK_HcmDgtlset600 ctcK_HcmDgtlset700
digitalAttributes	This field identifies the features on the digital station. It contains one or more of the following bitmasks: ctcM_HcmDgtlAttrDisplay ctcM_HcmDgtlAttrSpeaker ctcM_HcmDgtlAttrData ctcM_HcmDgtlAttrKeypadExp ctcM_HcmDgtlAttrISDN ctcM_HcmDgtlAttrTermAdptr

* otherEquipment

This element identifies the type of station if it is not an analog or digital station. It contains one of the following values:

ctcK_HcmOthrEquipGeneric
ctcK_HcmOthrEquipFax
ctcK_HcmOthrEquipOffPremise
ctcK_HcmOthrEquipExtVoiceMail

ctcK_HcmOthrEquipFictitious
ctcK_HcmOthrEquipPhantom
ctcK_HcmOthrEquipDataCommModule

- **hcmCallsQueued**

  This element is a 32-bit integer that contains the number of calls queued. It is supported for Siemens Hicom 300E switches only.

- **hcmStationFrwrdingParams**

  This element contains details of the call forwarding setting for the station. It is supported for Siemens Hicom 300E switches only, and is defined in the following fixed-format structure:

```
ctcHcmSetForwardElement{
       unsigned short      setableFwdType;
       ctcDeviceString     forwardDN
};
```

  The fields in this structure are:

  – **setableFwdType**

    This 16-bit field identifies the type of call forwarding set. It contains one of the following values:

    ctcK_HcmStnFwdTypeImmedIntOn
    ctcK_HcmStnFwdTypeImmedIntOff
    ctcK_HcmStnFwdTypeImmedExtOn
    ctcK_HcmStnFwdTypeImmedExtOff
    ctcK_HcmStnFwdTypeBusyNoAnsOn
    ctcK_HcmStnFwdTypeBusyNoAnsOff

  – **forwardDN**

    This null-terminated character string contains the number for the device. The maximum length for ctcDeviceString is specified by the literal ctcMaxDnLen defined in a CTC definitions file (see Section 1.5).

- **hcmSystemFrwrdingType**

  This element is a 16-bit integer that identifies the call forwarding setting defined on the switch. It is supported for Siemens Hicom 300E switches only, and contains one of the following values:

  ctcK_HcmPreconfSysFwdOn
  ctcK_HcmPreconfSysFwdOff

- **hcmForwardList**

  This element provides information about the call forwarding setting for a device that is not returned by ctcGetCallForward. It is supported for Siemens Hicom 300E switches only and is defined as the following fixed-format structure:

```
ctcHcmForwardList{
      unsigned short        count;
      ctcHcmForwardElement  forwards [10];
};
```

  The structure contains the following fields:

  – **count**

    This field specifies the number of ctcHcmForwardElement structures that contain data and are passed as part of the ctcHcmForwardList structure. The count value can be any number in the range 0 to 10.

  – **forwards**

    This field contains details of the call forwarding setting. It is defined as the following fixed-format structure:

```
ctcHcmForwardElement{
    ctcHcmForwardingType  hcmFwdType;
    unsigned short        stationFwdType;
    unsigned short        systemFwdType;
    ctcDeviceString       forwardDN;
 };
```

    The fields in this structure are:

    * **hcmFwdType**

      This field identifies which of the stationFwdType and systemFwdType fields contain data. It contains one of the following values:

Value	Description
ctcK_HcmFwdStation	Indicates that data is passed in the stationFwdType field.
ctcK_HcmFwdSystem	Indicates that data is passed in the systemFwdType field.

* **stationFwdType**

  This unsigned short can contain one of the following values:

  ctcK_HcmStnFwdTypeImmediateOn
  ctcK_HcmStnFwdTypeImmediateIntOn
  ctcK_HcmStnFwdTypeImmediateExtOn
  ctcK_HcmStnFwdTypeNoAnsOn
  ctcK_HcmStnFwdTypeBusyOn

* **systemFwdType**

  This unsigned short can contain one of the following values:

  ctcK_HcmSysFwdTypeImmediateOn
  ctcK_HcmSysFwdTypeImmediateIntOn
  ctcK_HcmSysFwdTypeImmediateExtOn
  ctcK_HcmSysFwdTypeNoAnswerOn
  ctcK_HcmSysFwdTypeNoAnswerIntOn
  ctcK_HcmSysFwdTypeNoAnswerExtOn
  ctcK_HcmSysFwdTypeBusyOn
  ctcK_HcmSysFwdTypeBusyIntOn
  ctcK_HcmSysFwdTypeBusyExtOn
  ctcK_HcmSysFwdTypeDoNotDisturbOn
  ctcK_HcmSysFwdTypeDoNotDisturbIntOn
  ctcK_HcmSysFwdTypeDoNotDisturbExtOn

* **forwardDN**

  This character string contains the number for the device. The maximum length for ctcDeviceString is specified by the literal ctcMaxDnLen defined in a CTC definitions file (see Section 1.5).

- **hcmRouteTrigger**

  This element specifies the setting for triggering all Route Control Groups (RCGs) configured on the switch. It is supported for Siemens Hicom 300E switches only and is defined as the following fixed-format structure:

```
ctcHcmRouteTrigger{
      boolean          triggerObjectSpecified;
      boolean          triggerAction;
      ctcDeviceString  triggerObject;
};
```

The structure contains the following fields:

– **triggerObjectSpecified**

This field specifies whether the trigger object is identified. It contains one of the following values:

ctcK_HcmTriggerObjectSpecified
ctcK_HcmTriggerObjectNotSpecified

– **triggerAction**

This field specifies the setting for triggering RCGs. It contains one of the following values:

ctcK_On
ctcK_Off

– **triggerObject**

This character string contains the number for the device. The maximum length for ctcDeviceString is specified by the literal ctcMaxDnLen defined in a CTC definitions file (see Section 1.5).

- **hcmRejectCall**

This element is a 32-bit integer that specifies the routing reference identifier when the request to route a call is rejected. It is supported for Siemens Hicom 300E switches only.

- **hcmRoutingEndCause**

This element specifies the reason for a call route to end. It is supported for Siemens Hicom 300E switches only and contains one of the following values:

Value	Description
ctcK_HcmRtEndCauseRtngTmrOrDlyRngbckTmrExprd	Routing Timer or Delay Ringback Timer expired
ctcK_HcmRtEndCauseCallerAbandonedCall	Caller hung up the call
ctcK_HcmRtEndCauseCallSuccessfullyRouted	Call successfully routed
ctcK_HcmRtEndCauseRdAbrtdDTRtSlctRsrcPrblm	Routing aborted due to route select resource problem

- **hcmAgentState**

This element provides additional information about the work mode for an agent that is not returned by ctcGetAgentStatus. It is supported for Siemens Hicom 300E switches only and is defined as the following fixed-format structure:

```
ctcHcmAgentState{
      unsigned short   agentState;
      unsigned short   agentIDLen;
      unsigned char    agentID [32];
      ctcDeviceString  agentGroup;
};
```

The structure contains the following fields:

– **agentState**

  This field specifies the work mode for the agent. It contains one of the following values:

  ctcK_HcmAgentStateNotReady
  ctcK_HcmAgentStateNull
  ctcK_HcmAgentStateReady
  ctcK_HcmAgentStateWorkNotReady
  ctcK_HcmAgentStateWorkReady

– **agentIDLen**

  This field specifies the length of the agent identifier passed in the agentID field.

– **agentID**

  This character string contains the identifier for the agent.

– **agentGroup**

  This character string contains the number for the agent group. The maximum length for ctcDeviceString is specified by the literal ctcMaxDnLen defined in a CTC definitions file (see Section 1.5).

• **ercEnterDTMF**

  This element contains DTMF tones so that, for an active call, an application can respond to systems that require DTMF tones as input. It is supported for Ericsson MD110 (BC9) switches only and is defined in the following fixed-format structure:

```
ctcErcEnterDTMF{
      char                 DTMFdigits [12];
      ctcErcConnectionId   connection;
};
```

The structure contains the following fields:

– **DTMFdigits**

This character string contains the DTMF tones to send.

– **connection**

This field contains details of the connection. It is defined as the following fixed-format structure:

```
ctcErcConnectionId{
      char           deviceId [12];
      unsigned int   callRefId;
};
```

The structure contains the following fields:

* **deviceId**

This character string contains the identifier for the device.

* **callRefId**

This 32-bit field contains the reference identifier for the call.

• **ercMessageDiversion**

This element sets or cancels diversion to an operator. It is supported for Ericsson MD110 (BC9) switches only and is defined in the following fixed-format structure:

```
ctcErcMessageDiversion{
      boolean               timeOrDateSpecified;
      char                  deviceId [12];
      boolean               mode;
      ctcErcDiversionType   diversionType;
      char                  ASN1TimeOrDate [5];
};
```

The structure contains the following fields:

– **timeOrDateSpecified**

This field contains one of the following values:

This value...	Specifies that...
ctcK_ErcTimeOrDateSpecified	A time or date is included in the ASN1TimeOrDate field.

This value...	Specifies that...
ctcK_ErcTimeOrDateNotSpecified	A time or date is *not* included in the ASN1TimeOrDate field.

– **deviceId**

This character string contains the identifier for the device.

– **mode**

This field specifies whether diversion is enabled or disabled. It contains one of the following values:

ctcK_On
ctcK_Off

– **diversionType**

This field contains one of the following values:

ctcK_ErcDivTypeMsgDiversion0
ctcK_ErcDivTypeMsgDiversion1
ctcK_ErcDivTypeMsgDiversion2
ctcK_ErcDivTypeMsgDiversion3
ctcK_ErcDivTypeMsgDiversion4
ctcK_ErcDivTypeMsgDiversion5
ctcK_ErcDivTypeMsgDiversion6
ctcK_ErcDivTypeMsgDiversion7
ctcK_ErcDivTypeMsgDiversion8
ctcK_ErcDivTypeMsgDiversion9

– **ASN1TimeOrDate**

If supplied, this field contains time or date information entered on the telephone keypad.

• **ercAccountCode**

This element contains a customer account code associated with either an active call or a new call. If a call is active, another line appearance on the set is used for the account code. If the device is idle, a line appearance is selected, the account code dialed, and the line remains ready for either manual dial or a ctcMakeCall request.

This element is supported for Ericsson MD110 (BC9) switches only and is defined in the following fixed-format structure:

```
ctcErcAccountCode{
      ctcErcAccountCodeChoice  choice;
      char                     accountCode[11];
      ctcErcConnectionId       connection;
};
```

The ctcErcAccountCode structure contains the following fields:

– **choice**

This field contains one of the following values:

This value...	Specifies that...
ctcK_ErcConnectionIdChosen	The account code is associated with an active call.
ctcK_ErcDeviceIdChosen	The account code is *not* associated with an active call.

– **accountCode**

This character string contains the account code.

– **connection**

This field contains details of the connection. It is defined as the following fixed-format structure:

```
ctcErcConnectionId{
      char           deviceId [12];
      unsigned int   callRefId;
};
```

The structure contains the following fields:

* **deviceId**

This character string contains the identifier for the device.

* **callRefId**

This 32-bit field contains the reference identifier for the call.

• **ercAuthCode**

This element contains a customer authorization code entered at a device. It is supported for Ericsson MD110 (BC9) switches only and is defined in the following fixed-format structure:

```
ctcErcAuthCode{
      char      deviceId[12];
      char      authCode[11];
};
```

The structure contains the following fields:

– **deviceId**

  This character string contains the identifier for the device.

– **authCode**

  This character string contains the authorization code.

• **ercPressProgKey**

  This element contains the number for a programmable key on an MD110 telephone to emulate the user pressing the key. Digits, the transfer key, conference key, and clear key are not supported. Note also that the key numbering on different types of digital telephone sets on the MD110 are not the same.

  This element is supported for Ericsson MD110 (BC9) switches only, and is defined in the following fixed-format structure:

```
ctcErcPressProgKey{
      char      deviceId[12];
      char      keyNumber[4];
};
```

  The structure contains the following fields:

– **deviceId**

  This character string contains the identifier for the device.

– **keyNumber**

  This character string contains the number of the key on the Digital Telephone Set (DTS) as defined on the MD110.

• **ercCancelCallback**

  This element is used to pass a request to cancel the previously set callback feature on a device. If the number for the called party is passed, only callback for that party is cancelled. If no number is included, all callback requests at the device are cancelled.

The element is supported for Ericsson MD110 (BC9) switches only and is defined in the following fixed-format structure:

```
ctcErcPressProgKey{
      boolean    callbackToCancelSpecified
      char       deviceId[12];
      char       callbacktoCancel[12];
};
```

The structure contains the following fields:

– **callbackToCancelSpecified**

This field contains one of the following values:

This value...	Specifies that...
ctcK_ErcCallbackToCancelSpecified	Callback to a specific called party is cancelled. The party is identified by the callbackToCancel field.
ctcK_ErcCallbackToCancelNotSpecified	All callback settings on the device are cancelled.

– **deviceId**

This character string contains the identifier for the device at which callback is invoked.

– **callbacktoCancel**

If supplied, this character string contains the number for the called party for which callback will be cancelled.

• **ercSetAssociateData**

This element is used to pass a dialable string with an external caller. It is supported for Ericsson MD110 (BC9) switches only and is defined in the following fixed-format structure:

```
ctcErcSetAssociateData{
      ctcErcConnectionId  connection
      char                associateData[21];
};
```

The structure contains the following fields:

– **connection**

This field contains details of the connection. It is defined as the following fixed-format structure:

```
ctcErcConnectionId{
      char          deviceId [12];
      unsigned int   callRefId;
};
```

The structure contains the following fields:

* **deviceId**

This character string contains the identifier for the device.

* **callRefId**

This 32-bit field contains the reference identifier for the call.

– **associateData**

This character string contains the data associated with the caller. The only valid characters in this string are 0 to 9 and A to F.

- **ercFwdACDGrp**

This element is used to request that an ACD group is forwarded to another destination or to cancel a forward ACD group request. It is supported for Ericsson MD110 (BC9) switches only and is defined in the following fixed-format structure:

```
ctcErcFwdACDGrp{
      char        deviceId[12];
      boolean     mode;
      char        acdDevice[12];
      char        fwdToDevice[12];
};
```

The structure contains the following fields:

– **deviceId**

This character string contains the identifier for the device.

– **mode**

This field specifies whether forwarding is enabled or disabled. It contains one of the following values:

ctcK_On
ctcK_Off

– **acdDevice**

This character string contains the number for the ACD group to be forwarded.

– **fwdToDevice**

This character string contains the number for the destination device.

- **ercEvtAssociateData**

This element contains the dialable string associated with an external caller.

- **ercFreeQueuePos**

This element is used to release a saved queue position for a call deflected from the queue. It is defined as the following fixed-format structure:

```
ctcErcConnectionId{
      char           deviceId [12];
      unsigned int   callRefId;
};
```

The structure contains the following fields:

– **deviceId**

This character string contains the identifier for the queue.

– **callRefId**

This 32-bit field contains the reference identifier for the deflected call.

B.15.4 Private Data Routines

The following pages describe each of the private data routines supported.

ctcCstaEscape
Exchange Private Data With the Switch

Format in C

*unsigned int **ctcCstaEscape*** *(ctcChanId channel,*
*ctcPrivateDataArray *privateDataArray)*

Description

This routine enables you to send and receive private data from the switch. The type of data you can exchange depends on the switch you are using. For more information, refer to your switch documentation or consult your switch support contact.

Differences Between ctcCstaEscape, ctcCstaSetPrivateData, and ctcCstaGetPrivateData

Both ctcCstaEscape and ctcCstaSetPrivateData enable you to send private data to the switch. However, with ctcCstaSetPrivateData, the data is stored on the CTC server and sent to the switch when you next call a CTC routine for that channel. ctcCstaEscape enables you to send private data to the switch without associating it with another CTC routine.

Similarly, ctcCstaEscape enables you to receive private data from the switch when the routine returns. ctcCstaGetPrivateData retrieves private data stored on the CTC server. The switch sends the data to the CTC server with the result of the previous routine called for the channel.

For more information, refer to the descriptions of the ctcCstaSetPrivateData and ctcCstaGetPrivateData routines in this appendix.

Alcatel, Ericsson MD110, and Hicom 300E Functions

A number of ctcCstaEscape values are specific to Alcatel 4400, Ericsson MD110, and Hicom 300E switches. Using these values, you can determine the type of private data you exchange with these switches. This enables you to tailor your application so that it provides switch-specific functions.

Dialogic recommends you work with your switch support contact to ensure that your application makes effective use of the flexibility offered by the ctcCstaEscape routine.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier returned by ctcAssign for the device in use.

The ctcChanId datatype is defined in a CTC definitions file (see Section 1.5).

privateDataArray

type: **ctcPrivateDataArray**
access: **read and write**
mechanism: **by reference**

This argument is the address of a fixed-format structure that contains the private data. The structure is formatted as follows:

```
ctcPrivateDataArray {
        ctcPrivateData privData [3];
};
```

ctcPrivateDataArray is an array of three ctcPrivateData structures. For details, refer to Section B.15.3.

Example

The following example shows how to call ctcCstaEscape to pass a single piece of raw data to the switch. It also shows how to use ctcCstaEscape to receive any private data at the switch. This private data is returned when ctcCstaEscape returns.

```
{
unsigned int status;
ctcPrivateDataArray privDataArray;
ctcPrivateData *privData;
char myData[] = “My private data”;

privData = &privDataArray.privData[0];
privData->privDataType = ctcK_PrivRaw;
privData->privDataValue.raw.datalen = strlen(myData);
strcpy(privData->privDataValue.raw.data, myData);
/*
 * Only one ctcPrivateData structure contains private data.
 * For the other two structures in ctcPrivateDataArray, set
 * the ctcPrivateDataType to ctcK_privNone.
 * /
privDataArray.privData[1].privDataType = ctcK_PrivNone;
```

```
privDataArray.privData[2].privDataType = ctcK_PrivNone;

status = ctcCstaEscape (channel, &privDataArray);
If (status == ctcSuccess);
    {
    /* To receive any private data that may be at the switch
     */

    for (i = 0; i < 3; i++)
        {
        privData = &privDataArray.privData[i];
        switch (privData->privDataType)
            {
            case ctcK_PrivRaw:
                   /*
                    * Handle as appropriate
                    */
            break;
            case ctcK_PrivAlcAcdWaitingTime:
                   /*
                    * Handle as appropriate
                    */
            break;
                   /*
                    * Add handlers for other private data
                    * as appropriate
                    */
            default:
            break;
            }
        }
    }
return (status);
}
```

ctcCstaGetPrivateData
Get Private Data Stored on the CTC Server

Format in C

unsigned int ***ctcCstaGetPrivateData*** *(ctcChanId channel,*
*ctcPrivateDataArray *privateDataArray)*

Description

This routine retrieves private data sent by the switch with the result to a CTC request. Private data can be sent by the switch whenever it sends a result message to CTC. For example, when it responds to another CTC routine such as ctcHangupCall.

The data is not sent to your application when the routine returns, but is stored on the CTC server. You call ctcCstaGetPrivateData to retrieve the data.

The data content is specific to CSTA switches. For details, refer to your switch documentation or consult your switch support contact.

Use this routine after calling a CTC routine that you know will generate private data from the switch. CTC cannot provide notification that private data is stored on the CTC server for the channel.

> **Note**: If data is stored on the switch and you call another routine before ctcCstaGetPrivateData, the private data is lost.

For details of the differences between ctcCstaGetPrivateData and ctcCstaEscape, refer to the description of ctcCstaEscape.

Alcatel, Ericsson MD110, and Hicom 300E Functions

A number of ctcCstaGetPrivateData values are returned by Alcatel, Ericsson MD110 (BC9), and Hicom 300E switches only.

If you are using one of these switches, Dialogic recommends you contact your switch support representative for full details of the private data that can be returned by the ctcCstaGetPrivateData routine.

ctcCstaGetPrivateData

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier returned by ctcAssign for the device in use.

The ctcChanId datatype is defined in a CTC definitions file (see Section 1.5).

privateDataArray

type: **ctcPrivateDataArray**
access: **write**
mechanism: **by reference**

This argument is the address of a fixed-format structure that receives the private data. The structure is formatted as follows:

```
ctcPrivateDataArray {
        ctcPrivateData privData [3]
};
```

ctcPrivateDataArray is an array of three ctcPrivateData structures. For details of the fields in this structure, refer to Section B.15.3.

ctcCstaGetPrivateEventData
Get Private Data Sent With Events

Format in C

unsigned int ***ctcCstaGetPrivateEventData*** *(ctcChanId channel,*
ctcPrivateDataArray privateDataArray)

Description

This routine retrieves private data associated with the last event returned on the channel.

CTC notifies you that private data has been sent by the switch by returning a value in the privateData field of the ctcEventData structure. Refer to the description of ctcGetEvent in Chapter 2 for details.

To retrieve the private data, you must call ctcCstaGetPrivateEventData **before** you repost ctcGetEvent. If you do not, the private data is lost.

Alcatel, Ericsson MD110, and Hicom 300E Functions

A number of ctcCstaGetPrivateEventData values are returned by Alcatel, Ericsson MD110 (BC9), and Hicom 300E switches only.

If you are using one of these switches, Dialogic recommends you contact your switch support representative for full details of the private data that can be returned by the ctcCstaGetPrivateEventData routine.

Arguments

channel

type:	**ctcChanId**
access:	**read only**
mechanism:	**by value**

This argument is a ctcChanId datatype that contains the channel identifier returned by ctcAssign for the device in use.

The ctcChanId datatype is defined in a CTC definitions file (see Section 1.5).

privateDataArray

type: **ctcPrivateDataArray**
access: **write**
mechanism: **by reference**

This argument is the address of a fixed-format structure that receives the private data. The structure is formatted as follows:

```
ctcPrivateDataArray {
     ctcPrivateData privData [3];
};
```

For details of the fields in this structure, refer to Section B.15.3.

ctcCstaGetPrivateRouteData
Get Private Data Sent With Route Requests

Format in C

```
unsigned int ctcCstaGetPrivateRouteData
                    (ctcChanId            channel,
                     ctcPrivateDataArray  *privateDataArray)
```

Description

This routine retrieves private data associated with the last route request sent by the switch for the assigned channel.

CTC notifies you that private data has been sent by the switch by returning a value in the privateData field of the ctcRouteData structure. Refer to the description of ctcGetRouteQuery in Chapter 2 for details.

To retrieve the private data, you must call ctcCstaGetPrivateRouteData **before** you repost ctcGetRouteQuery. If you do not, the private data is lost.

Alcatel, Ericsson MD110, and Hicom 300E Functions

A number of ctcCstaGetPrivateRouteData values are returned by Alcatel, Ericsson MD110 (BC9), and Hicom 300E switches only.

If you are using one of these switches, Dialogic recommends you contact your switch support representative for full details of the private data that can be returned by the ctcCstaGetPrivateRouteData routine.

Arguments

channel

type:	**ctcChanId**
access:	**read only**
mechanism:	**by value**

This argument is a ctcChanId datatype that contains the channel identifier returned by ctcAssign for the device in use.

The ctcChanId datatype is defined in a CTC definitions file Section 1.5.

privateDataArray

type: **ctcPrivateDataArray**
access: **write**
mechanism: **by reference**

This argument is the address of a fixed-format structure that receives the private data. The structure is formatted as follows:

```
ctcPrivateDataArray {
     ctcPrivateData privData [3];
};
```

For details of the fields in this structure, refer to Section B.15.3.

ctcCstaSetPrivateData
Set Private Data

Format in C

unsigned int ***ctcCstaSetPrivateData***
(ctcChanId channel,
*ctcPrivateDataArray *privateDataArray)*

Description

This routine sends private data to the switch for the specified channel. This data is stored on the CTC server until you call a CTC routine for that channel that sends a message to the switch. For details of the CTC routines that provide this information to the switch, contact Dialogic and your switch manufacturer.

For details of the differences between ctcCstaGetPrivateData and ctcCstaEscape, refer to the description of ctcCstaEscape.

Alcatel, Ericsson MD110, and Hicom 300E Functions

A number of ctcCstaSetPrivateData values can be used to send private data to Alcatel, Ericsson MD110 (BC9), and Hicom 300E switches.

If you are using one of these switches, Dialogic recommends you contact your switch support representative for full details of the private data that can be sent using the ctcCstaSetPrivateData routine.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier returned by ctcAssign for the device in use.

The ctcChanId datatype is defined in a CTC definitions file (see Section 1.5).

ctcCstaSetPrivateData

privateDataArray

type: **ctcPrivateDataArray**
access: **read**
mechanism: **by reference**

This argument is the address of a fixed-format structure that sends the private data. The structure is formatted as follows:

```
ctcPrivateDataArray {
     ctcPrivateData privData [3];
};
```

For details of the fields in this structure, refer to Section B.15.3.

B.16 Condition Values Returned

This section lists the condition values returned by a switch supporting the CSTA protocol. See Chapter 3 for definitions of these condition values. The specific condition values that can be returned for individual routines depend on your switch.

ctcBadObjState
ctcConfMemberLimEx
ctcExTrunkLimEx
ctcInvalidDest
ctcInvalidFeature
ctcInvAllocState
ctcInvCalledDevice
ctcInvCallIdentifier
ctcInvCallingDevice
ctcInvConnIdActCall
ctcInvConnIdentifier
ctcInvCrossRefId
ctcInvDevIdentifier
ctcInvForwardingDest
ctcInvObjectType
ctcInvPrivateData
ctcNetBusy
ctcNetOutOfServ
ctcNoActiveCall
ctcNoCallToAnswer
ctcNoCallToClear
ctcNoCallToComplete
ctcNoConnToClear
ctcNoHeldCall
ctcNoPrivateData
ctcObjectNotKnown
ctcObjMonLimEx
ctcOpGeneric
ctcOptNotSup
ctcOutstandReqLimEx
ctcOverallMonLimEx
ctcPacErr
ctcPerfGeneric
ctcPerfLimEx
ctcPrivateCstaErr
ctcPrivViolCalledDev
ctcPrivViolCallingDev
ctcPrivViolSpecDev
ctcReqIncomWithObj

ctcResourceBusy
ctcResOutOfServ
ctcSealErr
ctcSecGeneric
ctcSecurityViol
ctcSeqNumErr
ctcServiceBusy
ctcStGeneric
ctcSubsGeneric
ctcSwitchDisabled
ctcSwitchEnabled
ctcSwitchInit
ctcSwitchOverRel
ctcSysGeneric
ctcTimeStampErr
ctcUnspecCstaErr
ctcValueOutOfRange

The following additional condition values can be returned by switches supporting CSTA Phase II:

ctcInvAccountCode
ctcInvApplCorrelator
ctcInvAuthCode
ctcReqIncomWithCallingDev
ctcReqIncomWithCalledDev

C

Features Specific to the Lucent DEFINITY Generic

This appendix identifies aspects of CTC that are specific to a link with Lucent DEFINITY Generic 3 (G3) switches that CTC supports at this version. It indicates which routines are supported by these switches, and notes any differences between the supported routines and the descriptions in Chapter 2.

Appendix A lists the CTC features and functions Dialogic believes will be available with all switches supported by future versions of CTC. If you need to write applications that will work with more than one switch, Appendix A.

C.1 CTC Functions Supported by DEFINITY G3 Switches

Table C–1 indicates which routines are supported by the DEFINITY G3 switches. If a routine is listed as *Supported as noted*, all limitations are documented in this appendix.

Table C–1 CTC Routines for DEFINITY G3 Switches

Name of Routine	Support
ctcAddMonitor	Supported fully
ctcAnswerCall	Supported fully
ctcAssign	Supported as noted in Section C.3
ctcAssociateData	Not supported
ctcCancelCall	Supported as noted in Section C.4
ctcConferenceJoin	Supported fully
ctcConsultationCall	Supported fully
ctcDeassign	Supported fully
ctcDeflectCall	Supported as noted in Section C.5†
ctcErrMsg	Supported fully
ctcGetAgentStatus	Supported as noted in Section C.6
ctcGetCallForward	Supported as noted in Section C.7
ctcGetChannelInformation	Supported as noted in Section C.8
ctcGetDoNotDisturb	Supported fully
ctcGetEvent	Supported as noted in Section C.9
ctcGetMessageWaiting	Supported fully
ctcGetMonitor	Supported fully
ctcGetRouteQuery	Supported as noted in Section C.10
ctcGetRoutingEnable	Not supported
ctcHangupCall	Supported as noted in Section C.11
ctcHoldCall	Supported fully
ctcMakeCall	Supported as noted in Section C.12

†Not supported for DEFINITY G3 switches running ASAI Generic 3 Version 3 software. See Section C.2.

Table C–1 CTC Routines for DEFINITY G3 Switches (Continued)

Name of Routine	Support
ctcMakePredictiveCall	Supported as noted in Section C.13
ctcPickupCall	Not supported
ctcReconnectHeld	Supported fully
ctcRemoveMonitor	Supported fully
ctcRespondToInactive	Not supported
ctcRespondToRouteQuery	Supported as noted in Section C.14
ctcRetrieveHeld	Supported fully
ctcSendDTMF	Supported as noted in Section C.2†
ctcSetAgentStatus	Supported as noted in Section C.15
ctcSetCallForward	Supported as noted in Section C.16
ctcSetDoNotDisturb	Supported as noted in Section C.17
ctcSetMessageWaiting	Supported fully
ctcSetMonitor	Supported fully
ctcSetRoutingEnable	Not supported
ctcSingleStepTransfer	Not supported
ctcSnapshot	Supported as noted in Section C.18†
ctcSwapWithHeld	Supported fully
ctcTransferCall	Supported fully
ctcWinGetEvent	Supported as noted in Section C.9
ctcWinGetRouteQuery	Supported as noted in Section C.10

†Not supported for DEFINITY G3 switches running ASAI Generic 3 Version 3 software. See Section C.2.

Sections C.2 to C.17 contain information you require for writing a CTC application that uses a link to a DEFINITY G3 switch.

This information includes details of the technical distinctions to note when using the routines listed as *Supported as noted* in Table C–1. If you write an application that uses these features, you will probably have to modify it to work with other CTC-compatible switches.

C.2 Lucent DEFINITY Software

CTC uses the ASAI (Adjunct/Switch Application Interface) protocol to communicate with DEFINITY G3 switches.

Table C–1 shows support for CTC routines if the Lucent DEFINITY switch is running ASAI Generic 3 Version 4 (G3V4) or later.

If the switch is running ASAI Generic 3 Version 3 (G3V3):

- The following CTC routines are not supported:

  ctcDeflectCall
  ctcSendDTMF
  ctcSnapshot

- The ctcK_AgentLoggedOn event is not supported.

C.3 ctcAssign

This section describes operating differences and points to note when you use ctcAssign with the DEFINITY G3.

C.3.1 Supported Devices

You can assign a channel to the following devices:

- Voice sets (telephones)
- Groups (ACD splits and skill-based hunt groups on DEFINITY switches that support Expert Agent Selection)
- Route points
- Monitor channels

C.3.2 Assigning a Channel to a Route Point

To assign a channel to a route point, specify the Vector Directory Number (VDN) of the call vector that includes an adjunct routing point. Route point is a CTC term for a VDN.

C.3.3 Devices and Supported Routines

Table C–2 shows the routines supported for each type of device. A cross (X) indicates that the routine is supported.

Table C–2 Routines Supported for DEFINITY G3 Devices

CTC Routine	Voice Set	Group†	Route Point	Monitor Channel
ctcAddMonitor				X
ctcAnswerCall	X			
ctcAssign	X	X	X	X
ctcCancelCall	X			
ctcConferenceJoin	X			
ctcConsultationCall	X			
ctcDeassign	X	X	X	X
ctcDeflectCall	X			
ctcErrMsg	X	X	X	X
ctcGetAgentStatus	X			
ctcGetCallForward	X			
ctcGetChannelInformation	X	X	X	X
ctcGetDoNotDisturb	X			
ctcGetEvent	X	X	X	X
ctcGetMessageWaiting	X			
ctcGetMonitor	X	X	X	
ctcGetRouteQuery			X	
ctcHangupCall	X	X		
ctcHoldCall	X			
ctcMakeCall	X			
ctcMakePredictiveCall		X	X	
ctcReconnectHeld	X			
ctcRemoveMonitor				X
ctcRespondToRouteQuery			X	
ctcRetrieveHeld	X			

†Channels assigned to ACD splits or skill-based hunt groups.

Table C–2 Routines Supported for DEFINITY G3 Devices (Continued)

CTC Routine	Voice Set	Group†	Route Point	Monitor Channel
ctcSetAgentStatus	X			
ctcSetCallForward	X			
ctcSetDoNotDisturb	X			
ctcSetMessageWaiting	X			
ctcSetMonitor	X	X	X	
ctcSendDTMF	X			
ctcSnapshot	X	X		
ctcSwapWithHeld	X			
ctcTransferCall	X			
ctcWinGetEvent	X	X	X	
ctcWinGetRouteQuery			X	

†Channels assigned to ACD splits or skill-based hunt groups.

C.4 ctcCancelCall

This section describes operating differences when you use ctcCancelCall with the DEFINITY G3. For a full description of this routine, refer to Chapter 2.

C.4.1 Device State

When the ctcCancelCall routine disconnects a consultation call, the assigned device returns to the idle state, rather than the initiate state.

C.5 ctcDeflectCall

This section describes operating differences and points to note when you use ctcDeflectCall with the DEFINITY G3. For a full description of this routine, refer to Chapter 2.

C.5.1 Required Software

The ctcDeflectCall routine is supported by switches running ASAI G3V4 or later.

C.5.2 Supplying Application Data

The applicationData argument for this routine is not supported. Pass a zero-length, NUL-terminated string with this argument.

C.6 ctcGetAgentStatus

This section describes operating differences when you use ctcGetAgentStatus with the DEFINITY G3. For a full description of this routine, refer to Chapter 2.

C.6.1 Supplying Agent Data

If the switch you are using does not support Expert Agent Selection (EAS), you must supply the device number of a group (an ACD split) with the agentData argument. For DEFINITY G3 switches, the access for this argument is **read and write**.

The agentData argument usually returns information from the CTC server (see the description of this ctcGetAgentStatus argument in Chapter 2). However, because the DEFINITY G3 allows an agent to log in to more than one ACD split at a time, your application must supply a device number so that the correct ACD split can be identified.

If the switch you are using is set up to support EAS, you do not need to specify information with the agentData argument.

C.7 ctcGetCallForward

This section describes operating differences when you use ctcGetCallForward with the DEFINITY G3. For a full description of this routine, refer to Chapter 2.

C.7.1 Call Forward Modes

The only forwardMode value returned is ctcK_CfAll. Other forwardMode values are not supported.

C.8 ctcGetChannelInformation

This section describes operating differences and point to note when you use ctcGetChannelInformation with the DEFINITY G3. For a full description of this routine, refer to Chapter 2.

C.8.1 Line Types

The following values can be returned in the lineType field of the ctcChanData structure:

ctcK_LineACD
ctcK_LineMonitorChannel
ctcK_LineRoutePoint
ctcK_LineVoiceSet

The values ctcK_LineDataSet and ctcK_LineTrunk are not supported.

C.8.2 Set Types

The values returned in the setType field of the ctcChanData structure are:

ctcK_AsaiSetAnalog
ctcK_AsaiSetProprietary
ctcK_AsaiSetBasicRateISDN

C.8.3 Switch-Specific Support

If you are using the DEFINITY G3-specific routine, ctcAsaiGetAcdStatus (see Section C.19), the following value can be returned in the switchSpecificSupport field for the DEFINITY G3:

ctcM_AsaiGetAcdStatus

C.9 ctcGetEvent and ctcWinGetEvent

This section describes operating differences and points to note when you use ctcGetEvent or ctcWinGetEvent with the DEFINITY G3. For a full description of these routines, refer to Chapter 2.

C.9.1 Fields Used in the ctcEventData Structure

Table C–3 shows which fields are used in the ctcEventData structure for DEFINITY G3 switches. If Table C–3 specifies that the field is *Not supported*, CTC always returns null data for that field.

Table C–3 Event Information Supported by DEFINITY Switches

Field	Support
refId	See Chapter 2
netCallId	Not supported
oldRefId	See Chapter 2

Table C–3 Event Information Supported by DEFINITY Switches (Continued)

Field	Support
oldNetCallId	Not supported
state	See Section C.9.10
event	See Sections C.9.2, C.9.3, C.9.4, and C.9.6
eventQualifier	See C.9.11
type	See C.9.9
otherPartyType	See C.9.7
otherPartyQualifier	See C.9.8
otherParty	See Sections D.8.12 and C.9.13
otherPartyTrunk	See Chapter 2
otherPartyGroup	See Chapter 2
thirdPartyType	See Section C.9.7
thirdPartyQualifier	See Section C.9.8
thirdParty	See Sections D.8.12 and C.9.13
thirdPartyTrunk	See Chapter 2
thirdPartyGroup	See Chapter 2
calledPartyType	See Section C.9.7
calledPartyQualifier	See Section C.9.8
calledParty	See Section C.9.13
calledPartyTrunk	See Chapter 2
calledPartyGroup	See Chapter 2
applicationData	See Section C.9.14
monitorParty	See Chapter 2
nestedMonitorChannel	See Chapter 2
agentMode	See Section C.9.6
agentId	See Chapter 2
agentGroup	See Chapter 2
agentData	Not supported

Table C–3 Event Information Supported by DEFINITY Switches (Continued)

Field	Support
logicalAgent	See Section C.9.6
dtmfDigits	Not supported
originatingPartyType	Not supported
originatingPartyQualifier	Not supported
originatingParty	Not supported
originatingPartyTrunk	Not supported
originatingPartyGroup	Not supported
secOldRefId	Not supported
callsQueued	Not supported
accountInfo	Not supported
timeStamp	See Section C.9.15
privateData	Not supported

C.9.2 Events Not Returned

The following events are not supported by the DEFINITY G3:

ctcK_AgentModeChange
ctcK_BackInService
ctcK_CallInformation
ctcK_DestInvalid
ctcK_DestNotObtainable
ctcK_Error
ctcK_OffhookPrompt
ctcK_OutOfService
ctcK_Private

C.9.3 Information Returned for Channels Assigned to Route Points or Groups

If you assign a channel to a route point (VDN) or group (an ACD split or skill-based hunt group), the ctcGetEvent and ctcWinGetEvent routines return information about telephone activity for that route point or group. You can also receive information on a call originally active on the route point or group queue that has then been switched to another device.

A call can be switched to another device if:

- The call has been answered by an agent of the group.

- The call has been sent to a coverage path for the group.
- The call has been routed.

C.9.4 Events Returned for Channels Assigned to Groups

For channels assigned to groups, the following events are supported:

ctcK_AgentLoggedOn
ctcK_AgentLoggedOut

The agent can log on or log out either by pressing a key on the telephone keypad or by using ctcSetAgentStatus.

C.9.5 Event Returned for Monitored Groups

DEFINITY G3 switches support an extra monitoring event for groups (queues or ACD splits). The ctcK_DestinationChanged event is returned when a call has been delivered to an extension or agent of another ACD split.

C.9.6 Agent Events

Only the following agent events are supported:

- ctcK_AgentLoggedOff for switches running ASAI G3V3 software.
- ctcK_AgentLoggedOn and ctcK_AgentLoggedOff for switches running ASAI G3V4 or later software.

The following table shows agent information returned in ctcEventData fields for these events.

Event	Field	Contents
ctcK_AgentLoggedOn		
	agentMode	ctcK_AgentLogin
	agentId	Identifier for the agent
	agentGroup	DN for the group
	agentData	Null
	logicalAgent	DN for a logical agent†

†If the switch you are using is set up to support Expert Agent Selection (EAS), information is also returned in the logicalAgent field.

Event	Field	Contents
ctcK_AgentLoggedOff		
	agentMode	ctcK_AgentLogout
	agentId	Identifier for the agent
	agentGroup	DN for the group
	agentData	Null
	logicalAgent	DN for a logical agent†

†If the switch you are using is set up to support Expert Agent Selection (EAS), information is also returned in the logicalAgent field.

C.9.7 Party Type Information

ctcK_Dn is the only value returned by DEFINITY G3 switches in the following party fields:

otherPartyType
thirdPartyType
calledPartyType

The values ctcK_LineId (for CLID) and ctcK_Dnis (for DNIS) are not supported.

C.9.8 Party Qualifier

The DEFINITY G3 switches can return additional party information in the following fields:

otherPartyQualifier
thirdPartyQualifier
calledPartyQualifier

Each field can contain one of the following literals:

ctcK_AlertingDevice
ctcK_AnsweringDevice
ctcK_CalledNumber
ctcK_CallingDevice
ctcK_ConfController
ctcK_HoldingDevice
ctcK_NewRedirection
ctcK_ReleasingDevice
ctcK_RetrievingDevice
ctcK_TransferringDevice

C.9.9 Call Types

If your Lucent DEFINITY switch is running ASAI G3V5, CTC can return the values shown in Table C–4 in the type field of the ctcEventData structure.

If your switch is running an earlier version of ASAI, CTC returns null data only.

Table C–4 DEFINITY Call Types

Value	Description
ctcK_AsaiIdentifiedLine	The line is identified. There is no special treatment for the call.
ctcK_AsaiMultipartyNoANI	The switch cannot provide ANI for the call.
ctcK_AsaiANIfailure	ANI failed for the call.
ctcK_AsaiHotelMotel	Automatic room identification was not supplied with the hotel or motel DN.
ctcK_AsaiOperatorHandlingRequired	Special operator handling is required.
ctcK_AsaiAIOD	Automatic Identification of Outbound DNs. The DN (for example, extension number) was supplied by the PBX.
ctcK_AsaiCoinOrNonCoin	The line status is unknown.
ctcK_Asai800ServiceCall	The call is an 800 service call.
ctcK_AsaiCoinCall	The call was made from a coin-operated device.
ctcK_AsaiPrisonInmateService	The call is a prison service call.
ctcK_AsaiIntercept30	Intercepted call (30).
ctcK_AsaiIntercept31	Intercepted call (31).
ctcK_AsaiIntercept32	Intercepted call (32).
ctcK_AsaiTelcoOperatorHandled	The call was handled by a local telecommunications company operator.
ctcK_AsaiOutWATS	The call is an outbound Wide Area Telecommunications Service (WATS) call.
ctcK_AsaiTRSstationPaid	The Telecommunications Relay Service (TRS) station paid.
ctcK_AsaiType1Cellular	The call is a Type 1 cellular call.
ctcK_AsaiType2Cellular	The call is a Type 2 cellular call.
ctcK_AsaiRomerCellular	The call is a roamer cellular call.

Table C–4 DEFINITY Call Types (Continued)

Value	Description
ctcK_AsaiTRSfromHotelMotel	TRS from a hotel or motel.
ctcK_AsaiTRSfromRestrictedLine	TRS from a restricted line.
ctcK_AsaiPrivatePaystation	The call is from a private paystation.
ctcK_AsaiPrivateVirtualNetwork	The call was made over a private virtual network.

Note that the values supported may be returned for certain geographical regions only. CTC may also return additional values sent by the switch. To interpret these values, or to obtain more information about the values in Table C–4, refer to Bellcore's *Local Exchange Routing Guide* (TR-EOP-000085). For details, contact Bellcore.

C.9.10 Call Events and States

Table C–5 shows the possible device states that can be returned for each **call** event.

Table C–5 Call Events and States Returned

Event	Description	States
ctcK_DestBusy	The dialed destination is busy.	Fail
ctcK_DestChanged	The call from the assigned device was redirected to another destination.	Deliver Queued
ctcK_DestSeized	A call has been successfully dialed. If this call is external to the ACD, the network number has been verified and the outbound trunk seized. This does not indicate that the other end is actually ringing or answered.	Deliver Queued
ctcK_Diverted	A call has been diverted from this device to another.	Null
ctcK_InboundCall	A new call has arrived at the assigned device prior to routing.	Receive Queued
ctcK_Offhook	A new call has been made from the assigned device.	Initiate

Table C–5 Call Events and States Returned (Continued)

Event	Description	States
ctcK_OpAnswered	The other party has answered the call from the assigned device.	Active
ctcK_OpConferenced	The other party on the call has created a conference call.	Active
ctcK_OpDisconnected	The other party has hung up.	Null
ctcK_OpHeld	The other party is on hold.	Hold
ctcK_OpRetrieved	The held party has been retrieved.	Active
ctcK_Other	An event has occurred during the call (see Section C.9.11).	Fail
ctcK_TpAnswered	A call has been connected to this party.	Active Queued
ctcK_TpConferenced	This party has been connected in a conference call.	Active
ctcK_TpDisconnected	This party has been disconnected possibly because a call has been transferred, or because the telephone was off-hook too long.	Null Fail
ctcK_TpRetrieved	The held call has been retrieved by this party.	Active
ctcK_TpSuspended	This party has placed a call on hold.	Hold
ctcK_Transferred	The call has been transferred from another device, or this party transferred the call	Active Null

C.9.11 Call Event Qualifiers for DEFINITY G3 Switches

This section describes the DEFINITY G3 qualifiers for call events. Call events occur during the progress of a call and, along with call states, indicate the success or failure of calls on the monitored device. The qualifier can provide more information on the nature of the event.

CTC returns information about call events in the eventData structure returned by ctcGetEvent and ctcWinGetEvent. A DEFINITY G3 switch supplies more detailed information on events, and CTC returns this additional information in the eventQualifier field.

To determine which event has occurred, compare the value returned in the eventQualifier field with the literals listed as ctcK_Asai... in Table C–6. These literals define the possible qualifiers returned by a DEFINITY G3 switch and are supplied in the definitions file installed on your system.

Table C–6 Call Event Qualifiers for DEFINITY G3 Switches

Qualifier Literal	Description
ctcK_AsaiUserBusy	The number that has been called is busy.
ctcK_AsaiCallRejected	The caller is hearing the reorder tone. The call could not be placed.
ctcK_AsaiInvalidNumber	The called number does not fit in with the switch's number plan.
ctcK_AsaiNormUnspecified	Normal, unspecified qualifier.
ctcK_AsaiNoCircuit	No switch circuits are available.
ctcK_AsaiCallsBarred	The switch has barred that type of call.
ctcK_AsaiIncompatibleDest	Either something is wrong with the dialed digits or a data set has answered a switch-classified (predictive) call.
ctcK_AsaiUnspecified	The call has failed for an unspecified reason.
ctcK_AsaiTimedAnswer	Call answered on a non-ISDN trunk.
ctcK_AsaiVoiceEnergyAnswer	Classifier detected answer.
ctcK_AsaiTrunksNotAvailable	No trunks are currently available to make the call.
ctcK_AsaiQueuesFull	The group queues (ACD splits) are full.
ctcK_AsaiRemainsInQueue	The call is still queued at the original device even though other events may be returned.
ctcK_AsaiAnsweringMachine	Answering machine detected.
ctcK_AsaiInvalidCallId	Invalid call identifier.
ctcK_AsaiNormalClearing	The call has been cleared due to a transfer operation.

C.9.12 Mapping Qualifiers to Events

Table C–7 indicates which qualifiers may be generated by an event. Note that some events do not generate qualifiers and therefore are not included in this table.

Table C–7 DEFINITY Event Information Returned

Event	Qualifiers
ctcK_DestBusy	
	ctcK_AsaiTrunksNotAvailable ctcK_AsaiQueuesFull ctcK_AsaiNoCircuit ctcK_AsaiUserBusy
ctcK_DestSeized	
	ctcK_AsaiRemainsInQueue
ctcK_OpDisconnected	
	ctcK_AsaiUnspecified ctcK_AsaiNormalClearing
ctcK_Other	
	ctcK_AsaiUnspecified
ctcK_OpAnswered	
	ctcK_AsaiUnspecified ctcK_AsaiInvalidNumber ctcK_AsaiNormUnspecified ctcK_AsaiTimedAnswer ctcK_AsaiVoiceEnergyAnswer ctcK_AsaiAnsweringMachine
ctcK_TpAnswered	
	ctcK_AsaiUnspecified ctcK_AsaiNormUnspecified

Table C–7 DEFINITY Event Information Returned (Continued)

Event	Qualifiers
ctcK_TpDisconnected	
	ctcK_AsaiUnspecified ctcK_AsaiCallRejected ctcK_AsaiCallsBarred ctcK_AsaiIncompatibleDest ctcK_AsaiAnsweringMachine ctcK_AsaiInvalidCallId ctcK_AsaiInvalidNumber ctcK_AsaiNormalClearing

C.9.13 Party Information for Call Events

Table C–8 shows the DEFINITY G3 party information returned for supported **call** events.

Table C–8 DEFINITY Party Information for Call Events

Event	Field	Party Information	Explanation
ctcK_DestBusy			
	Other	Null	
	Third	Null	
	Called	Busy Party	
ctcK_DestChanged			
	Other	Ringing Party	
	Third	Null	New ACD DN if call has moved to another ACD.
	Called	Called Number	
ctcK_DestSeized			
	Other	Ringing Party	Can be called party for trunk seized or cut through.
	Third	Null	Can be calling party when a call to the monitored ACD is ringing at an agent.
	Called	Called Number	

Table C–8 DEFINITY Party Information for Call Events (Continued)

Event	Field	Party Information	Explanation
ctcK_Diverted			
	Other		Null
	Third	New group queue DN	
	Called		
ctcK_InboundCall			
	Other	Calling Party	
	Third	Null	
	Called	Called Number	
ctcK_Offhook			
	Other	Null	
	Third	Null	
	Called	Null or Called Number	The called number is returned after the number has been dialed.
ctcK_OpAnswered			
	Other	Answering Party	
	Third	Calling Party for ACD monitoring	
	Called	Called Party	
ctcK_OpConferenced			
	Other	Party joining conference or Conference Controller	
	Third	Conference Controller or Original Party	
	Called	Null	
ctcK_OpDisconnected			
	Other	Disconnecting Party	
	Third	Null	
	Called	Null	

Table C–8 DEFINITY Party Information for Call Events (Continued)

Event	Field	Party Information	Explanation
ctcK_OpHeld			
	Other	Holding Party	
	Third	Null	
	Called	Null	
ctcK_OpRetrieved			
	Other	Retrieving Device	When a held call is retrieved.
	Third	Null	
	Called	Null	
ctcK_Other			
	Other	Null	
	Third	Null	
	Called	Null	
ctcK_TpAnswered			
	Other	Party at Other End of Call	Calling party.
	Third	Null	
	Called	Called Number	
ctcK_TpConferenced			
	Other	New Party	
	Third	Original Party	
	Called	Null	
ctcK_TpDisconnected			
	Other	Null	
	Third	Null	
	Called	Null	

Table C–8 DEFINITY Party Information for Call Events (Continued)

Event	Field	Party Information	Explanation
ctcK_TpRetrieved			
	Other	Null	
	Third	Null	
	Called	Null	
ctcK_TpSuspended			
	Other	Null	
	Third	Null	
	Called	Null	
ctcK_Transferred			
	Other	New Party Connected	When the original party transfers a call in progress to a new party.
	Third	Transferring Party	
	Called	Null	

Table C–8 gives only DEFINITY party information. For other switch-specific tables (for example, Meridian), refer to the specific appendix. For a description of each event (such as ctcK_DestBusy), see Table 2–5.

C.9.14 Application Data for Events

DEFINITY G3 switches can return application data for the following events:

ctcK_InboundCall
ctcK_DestinationSeized
ctcK_DestinationChanged

C.9.15 Time Stamp

This field returns the date and time the event is processed on the CTC server, as described in Chapter 2.

Returning the date and time on the switch is not supported by DEFINITY switches.

C.10 ctcGetRouteQuery and ctcWinGetRouteQuery

This section describes operating differences and points to note when you use ctcGetRouteQuery or ctcWinGetRouteQuery with the DEFINITY G3. For a full description of these routines, refer to Chapter 2.

These routines present a call to the application so that the call can be routed. They are supported for channels assigned to Vector Directory Numbers (VDN) only. In CTC terms, VDNs are route points.

C.10.1 Fields Used in the ctcRouteData Structure

Table C–9 shows which fields in the ctcRouteData structure are used by DEFINITY G3 switches. If Table C–9 shows that a field is *Not supported*, CTC always returns null data for that field.

Table C–9 Route Information Supported by DEFINITY G3 Switches

Field	Support
routeId	As described in Chapter 2
refId	As described in Chapter 2
spare001	Not supported
otherPartyType	See Section C.10.2
otherParty	As described in Chapter 2
otherPartyTrunk	As described in Chapter 2
otherPartyGroup	As described in Chapter 2
thirdPartyType	Not supported
thirdParty	Not supported
thirdPartyTrunk	Not supported
thirdPartyGroup	Not supported
calledPartyType	See Section C.10.2
calledParty	As described in Chapter 2
calledPartyTrunk	Not supported
calledPartyGroup	Not supported
applicationData	As described in Chapter 2

Table C–9 Route Information Supported by DEFINITY G3 Switches (Continued)

Field	Support
dtmfDigits	See Section C.10.3
timeStamp	See Section C.10.4
privateData	Not supported

C.10.2 otherPartyType and calledPartyType Fields

ctcK_Dn is the only value returned by DEFINITY G3 switches in the otherPartyType and calledPartyType fields of the ctcRouteData structure.

C.10.3 DTMF Digits

If the call vector includes a “Collect Digits” stage, any DTMF digits entered will be returned in the dtmfDigits field.

C.10.4 Time Stamp

If the link is configured to return a time stamp from the CTC server, the timeStamp field contains the data and time the CTC server processed the route request. For more information, refer to the description of this field in Chapter 2.

If the link is configured to return a time stamp from the DEFINITY G3, this field contains null data. The DEFINITY G3 does not support time stamp information.

For more information about changing the configuration of a link, refer to the *CT-Connect Installation and Administration Guide* for your CTC server platform.

C.11 ctcHangupCall

This section describes operating differences and points to note when you use ctcHangupCall with the DEFINITY G3. For a full description of this routine, refer to Chapter 2.

C.11.1 Supported Devices

The ctcHangupCall routine is supported for channels assigned to groups (ACD splits) and route points (VDNs) only.

C.11.2 Disconnecting Calls Made With ctcMakePredictiveCall

The ctcHangupCall routine can be used on channels assigned to ACD split DNs and VDNs to disconnect calls made using the ctcMakePredictiveCall routine. These calls can be disconnected at any time before or after they are answered at the destination device. The callRefID of the call must be passed into ctcHangupCall.

Calls that arrive at an ACD split or VDN that have not been made using ctcMakePredictiveCall cannot be disconnected in this way and an attempt to do so results in a ctcInvCallIdentifier condition value.

C.12 ctcMakeCall

This section describes operating differences and points to note when you use ctcMakeCall with the DEFINITY G3. For a full description of this routine, refer to Chapter 2.

C.12.1 Supported Devices

The ctcMakeCall routine is supported for channels assigned to voice sets (telephones) only.

C.12.2 On-Hook Dialing

On-hook dialing is supported for all telephones.

C.12.3 Off-Hook Prompting

The off-hook prompting event is not supported for 2500 sets. The ctcMakeCall routine does not complete until the 2500 set is taken off hook. If this action is not taken within the time limit set on the switch, the routine returns with a ctcReqIncomWithCallingDev error.

C.13 ctcMakePredictiveCall

This section describes operating differences and points to note when you use ctcMakePredictiveCall with the DEFINITY G3. For a full description of this routine, refer to Chapter 2.

C.13.1 Supported Devices

The ctcMakePredictiveCall routine is supported for channels assigned to groups (ACD splits or skill-based hunt groups) and route points only.

C.13.2 Allocation

The following values are supported for the allocation argument:

Value	Description
ctcK_AllocDefault	Connection to an answer machine is not detected.
ctcK_AllocAMDAdmin	Call treatment on detecting an answer machine is specified in the switch administration.
ctcK_AllocAMDDrop	End the call if an answer machine is detected.
ctcK_AllocAMDConnect	Continue the call even if an answer machine is detected.

C.13.3 Number of Rings

Specify a value in the range 2 to 15 with the numberOfRings argument. If you pass the value zero, the switch uses the default of 15 rings.

C.14 ctcRespondToRouteQuery

This section describes operating differences and points to note when you use ctcRespondToRouteQuery with the DEFINITY G3. For a full description of this routine, refer to Chapter 2.

C.14.1 Dial-Ahead Digits

The newCalledNumber argument can be used to specify ASAI-provided dial-ahead digits with the new route for the call.

ASAI-provided dial-ahead digits allow you to associate DTMF digits with a call so that the call can be treated in a specific way at the new destination (for example, the digits can be used to indicate that a script is played). For more information about ASAI-provided dial-ahead digits, refer to the documentation provided with your switch.

To associate dial-ahead digits with the call, the newCalledNumber character string must contain:

- The number that identifies the new route for the call
- The hash character (#)
- The additional dial-ahead digits

Note that the character is used to separate the number that identifies the new route and the dial-ahead digits; it is not included in the digits sent to the switch.

C.15 ctcSetAgentStatus

This section describes operating differences and points to note when you use ctcSetAgentStatus with the DEFINITY G3. For a full description of this routine, refer to Chapter 2.

C.15.1 Supported Devices

The ctcSetAgentStatus routine is supported for channels assigned to voice sets (telephones) only.

C.15.2 Logging In Agents on EAS Switches

CTC supports Expert Agent Selection (EAS). If the switch you are using is set up to support EAS, you can specify information about the logical agent with the logicalAgent argument, and you do not need to provide agentGroup information.

The following example sequence sets up logical agent information on a DEFINITY G3 switch:

1. Assign a channel to the physical DN (the number of the telephone) at which the agent logs in.
2. Call ctcSetAgentStatus, specifying the following information:

For this argument...	Specify...
channel	Channel reference
agentMode	ctcK_AgentLogin
agentData	Password (if required) or the address of a zero-length character string
logicalAgent	DN for the logical agent
agentGroup	Address of a zero-length character string

3. Call ctcGetAgentStatus to query information about the state of the agent. No additional data is required: specify a zero-length character string with the agentGroup and agentData arguments (even if you supplied a password when the agent logged in).
4. Call ctcSetAgentStatus to change the agentMode information. No additional data is required: specify a zero-length character string with the agentGroup and agentData arguments (even if you supplied a password when the agent logged in).

C.15.3 Logging In Agents on Non-EAS Switches

If the switch you are using does not support EAS, you must supply the DN for a group (ACD split) with the agentGroup argument whenever you use ctcSetAgentStatus to change an agent's work mode.

Because a DEFINITY G3 switch allows an agent to log in to more than one ACD split at a time, the DN must be supplied so that the correct ACD split can be identified.

For example, to log in an agent, you specify the following ctcSetAgentStatus information:

For this argument...	Specify...
channel	Channel reference
agentMode	ctcK_AgentLogin
agentData	Password (if required) or the address of a zero-length character string
logicalAgent	Address of a zero-length character string
agentGroup	DN for the agent group (ACD split)

After logging in the agent, do not specify a password with the agentData argument for ctcSetAgentStatus. If you use ctcSetAgentStatus to change the work mode for the agent, supply the address of a zero-length character string with the agentData argument.

C.16 ctcSetCallForward

This section describes operating differences when you use ctcSetCallForward with the DEFINITY G3. For a full description of this routine, refer to Chapter 2.

C.16.1 Supported Settings

The only forwardMode value supported is ctcK_CfAll (set call forward on all calls).

C.17 ctcSetDoNotDisturb

This section describes operating differences when you use ctcSetDoNotDisturb with the DEFINITY G3. For a full description of this routine, refer to Chapter 2.

C.17.1 Busy Signal

When Do-Not-Disturb is set on, the call is deflected to the DEFINITY G3's coverage path. Therefore, when another party calls the assigned telephone, they do not hear a busy signal.

C.18 ctcSnapshot

This section describes operating differences and points to note when you use ctcSnapshot with the DEFINITY G3. For a full description of this routine, refer to Chapter 2.

C.18.1 Required Software

The ctcSnapshot routine is supported by switches running ASAI G3V4 or later.

C.18.2 Supported States

DEFINITY G3 switches return the following states only:

ctcK_ReceiveState
ctcK_InitiateState
ctcK_HoldState
ctcK_ActiveState

Note that if a call made from the assigned device is ringing at the destination device, the returned state is ctcK_ActiveState (and not ctcK_DeliverState as indicated in Table 2–4).

C.19 CTC Routine for the Lucent DEFINITY Switch

The following pages describe the Lucent DEFINITY-specific CTC routine that is provided as an extension to the standard CTC API. This routine is for CTC applications that will only be used with a Lucent DEFINITY switch.

To use this Lucent DEFINITY-specific CTC routine:

- You must assign the channel to an ACD split DN.
- When you use ctcAssign to assign the channel, you must specify the value ctcK_ASAI in the APIextensions field of the ctcAssignData structure. If you do not specify this value, you will not be able to use the routine for the assigned channel. Refer to the description of ctcAssign in Chapter 2 for more information.

ctcAsaiGetAcdStatus
Query the Status of an ACD Queue

Format in C

```
unsigned int ctcAsaiGetAcdStatus (ctcchanID     channel,
                                  unsigned int  *numberOfCalls,
                                  unsigned int  *numberOfLoggedInAgents,
                                  unsigned int  *numberOfAvailableAgents)
```

Description

The ctcAsaiGetAcdStatus routine returns information about the queue at an ACD split.

Using this routine, you can find out:

- The number of calls currently queued at the ACD split.
- The number of agents who are logged into the ACD split.
- The number of agents who are logged into the ACD split and are available to take calls.

Arguments

channel

type: **ctcChanID**
access: **read only**
mechanism: **by value**

This argument is a ctcChanID datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the device in use.

numberOfCalls

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This argument is the address of a 32-bit integer into which CTC writes the number of calls currently queued at the ACD split.

numberOfLoggedInAgents

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This argument is the address of a 32-bit integer into which CTC writes the number of agents who are logged into the ACD split.

numberOfAvailableAgents

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This argument is the address of a 32-bit integer into which CTC writes the number of agents who are logged into the ACD split and are available to take calls.

D

Features Specific to Nortel Meridian Switches

This appendix identifies aspects of CTC that are specific to a link with a Nortel Meridian 1™ or Meridian SL-1™ switch. It contains:

- Details of the standard CTC routines supported and any operating differences that are specific to Meridian 1 or Meridian SL-1 switches.

  If you need to write an application that will work with more than one switch, note the differences in this appendix. You should also refer to Appendix A for details of the CTC features and functions that are supported by all switches.
- Meridian-specific CTC routines provided as extensions to the CTC API. These routines are for CTC applications that will be used with a Meridian 1 or Meridian SL-1 only.

The procedures throughout this appendix apply to both the Meridian 1 and Meridian SL-1 switches. However, for simplicity, only the term Meridian is used throughout this appendix.

D.1 Meridian Switch Software

For your application to use the standard CTC routines supported by the Meridian (see Table D–2) and Meridian-specific routines (see Section D.16), your Meridian switch must be running:

- X11 Release 16 or later of the Meridian switch software.
- Release 3.0 or later of the Meridian Link software.

Dialogic recommends X11 Release 19 or later and Meridian Link software Release 4.0 or later. These releases support the CTC features shown in Table D–1.

Table D–1 Meridian Software and Supported CTC Features

Meridian Switch Software	Meridian Link Software	CTC Features
X11 Release 19 or later	Release 4.0	Call references (see Section D.3.2)
X11 Release 21 or later	Release 4.0	ctcSingleStepTransfer
X11 Release 22 or later	Release 5.0 (Co-Res 6.01 or later)	ctcHoldCall, ctcRetrieveHeld, additional call events (see Section D.8.7) and agent events (see Section D.8.6), and event time stamp information (see Section D.8.16)

D.2 Standard CTC Functions Supported by a Meridian Switch

Table D–2 indicates which of the standard CTC routines are supported by Meridian switches. These routines are described fully in Chapter 2. For details of the Meridian-specific routines available with CTC, refer to Section D.16.

Note that the latest versions of the Meridian switch software and Meridian Link software are required to support those routines shown as supported fully. See Section D.1.

Sections D.3 to D.15 point out the technical distinctions to note when writing applications that call the routines listed as *Supported as noted* in Table D–2. If you write an application that uses these features, or any of the Meridian-specific features described in Section D.16, you will have to modify it to work with other CTC-compatible switches.

Table D–2 CTC Routines and Meridian Switches

Name of Routine	Support
ctcAddMonitor	Supported fully.
ctcAnswerCall	Supported fully†‡.
ctcAssign	Supported as noted in Section D.4.
ctcAssociateData	Not supported.
ctcCancelCall	Not supported. For information about canceling calls, see Section D.5.
ctcConferenceJoin	Supported fully‡.
ctcConsultationCall	Supported as noted in Section D.6.
ctcDeassign	Supported fully.
ctcDeflectCall	Not supported.
ctcErrMsg	Supported fully.
ctcGetAgentStatus	Not supported.
ctcGetCallForward	Not supported.
ctcGetChannelInformation	Supported as noted in Section D.7.
ctcGetDoNotDisturb	Not supported.
ctcGetEvent	Supported as noted in Section D.8.
ctcGetMessageWaiting	Not supported.
ctcGetMonitor	Supported fully.
ctcGetRouteQuery	Supported as noted in Section D.9.
ctcGetRoutingEnable	Not supported.
ctcHangupCall	Supported fully†‡.
ctcHoldCall	Supported fully‡ with Meridian switch software X11 Release 22 or later and Meridian Link software Release 5.0 or later.
ctcMakeCall	Supported as noted in Section D.10.

†Not supported for channels assigned to 500 or 2500 sets.
‡Meridian switches do not use call reference identifiers associated with this routine. For more information, refer to Section D.3.2.

Table D–2 CTC Routines and Meridian Switches (Continued)

Name of Routine	Support
ctcMakePredictiveCall	Not supported.
ctcPickupCall	Not supported.
ctcReconnectHeld	Supported fully‡.
ctcRemoveMonitor	Supported fully.
ctcRespondToInactiveCall	Not supported.
ctcRespondToRouteQuery	Supported as noted in Section D.11.
ctcRetrieveHeld	Supported with Meridian switch software X11 Release 22 or later and Meridian Link software Release 5.0 or later.
ctcSendDTMF	Not supported.
ctcSetAgentStatus	Supported as noted in Section D.12.
ctcSetCallForward	Supported as noted in Section D.13.
ctcSetDoNotDisturb	Supported fully.
ctcSetMessageWaiting	Supported fully.
ctcSetMonitor	Supported fully.
ctcSetRoutingEnable	Not supported.
ctcSingleStepTransfer	Supported as noted in Section D.14.
ctcSnapshot	Not supported.
ctcSwapWithHeld	Not supported.
ctcTransferCall	Supported as noted in Section D.15.
ctcWinGetEvent	Supported as noted in Section D.8.
ctcWinGetRouteQuery	Supported as noted in Section D.9.

‡Meridian switches do not use call reference identifiers associated with this routine. For more information, refer to Section D.3.2.

D.3 Using CTC With Meridian Switches

Sections D.3.1 to D.3.3 contain general information for writing a CTC application that can be used with a Meridian switch.

D.3.1 Configuring the Meridian Switch

Before you can assign a channel to a voice set (terminal) or an ACD agent, the switch administrator must configure your Meridian switch so that:

- Each voice set or position ID to which you want to assign is defined as an associate set on the Meridian switch and configured for group 1 Meridian Link Unsolicited Status Messages (USMs):
  - For analog sets, use overlay program 10 (LD10) to set AST to YES and IAPG to group 1
  - For digital sets, use overlay program 11 (LD11) to set the AST to the appropriate key number and IAPG to group 1

  In addition, for position IDs, use overlay program 23 (LD23) to set ISAP to YES.
- The security option is set on. Use overlay program 17 (LD17) to set the SECU prompt to YES.

For more information about assigning channels, refer to Section D.4.

D.3.2 Call Reference Identifiers

This section contains information about call reference identifiers supplied to, or returned by, the Meridian switch.

callRefId, activeCallRefId, heldCallRefId Values

For the following routines, the Meridian switch does not use the value you provide with the callRefId, activeCallRefId, or heldCallRefId arguments:

ctcAnswerCall
ctcConferenceJoin
ctcConsultationCall
ctcHangupCall
ctcHoldCall
ctcReconnectHeld
ctcRetrieveHeld
ctcSingleStepTransfer
ctcTransferCall

However, to ensure that your application is compatible with other switches, Dialogic recommends that you pass the reference identifier for the call as returned by ctcGetEvent, ctcWinGetEvent, or ctcMakeCall.

If the Meridian is running X11 Release 18 or earlier of the Meridian switch software, it returns the value zero as the callRefId for ctcMakeCall.

newCallRefId Values

The values returned by the Meridian switch for newCallRefId are dependent on the release of switch software and link software running on the switch:

- If the Meridian is running switch software X11 Release 18 or earlier, it returns the value zero as the newCallRefId for the following routines:

  ctconferenceJoin
  ctcConsultationCall
  ctcSingleStepTransfer
  ctcTransferCall

- If the Meridian is running switch software X11 Release 19 or later and Meridian Link software Release 4.0 or later, it returns one of the following for newCallRefId:

  – For ctcConferenceJoin and ctcTransferCall, the call reference generated by the Meridian switch for the held call.

  – For ctcConsultationCall, the call reference generated by the Meridian switch for the new call.

  – For ctcSingleStepTransfer (supported with Meridian switch software X11 Release 22 or later and Meridian Link software Release 5.0 or later), the call reference generated by the Meridian for the original call.

D.3.3 Switch Overload

If you attempt to use a function (assign a channel or make a call, for example), and the Meridian switch is too busy to respond to the request, CTC returns one of the following errors:

- ctcSwitchOverImm—Switch overload is imminent
- ctcSwitchOverRch—Switch overload has been reached

Your application should wait and then repeat the request.

D.4 ctcAssign

This section describes operating differences and points to note when you use ctcAssign with a Meridian switch. For a full description of this routine, refer to Chapter 2.

D.4.1 Supported Devices

You can assign a channel to the following devices:

For this device...	Specify this deviceType value...
A voice set (*terminal* or *telephone*)	ctcK_Dn
An ACD agent	ctcK_Dn
An ACD group (*ACD queue DN*)	ctcK_Dn
A route point (*Controlled DN*)	ctcK_RoutePoint
A monitor channel	ctcK_MonitorChannel
A voice channel	ctcK_VoiceChannel (Meridian only)

Note that the device to which you assign must be a valid device. The value that you specify for deviceDN is not verified.

Table D–3 shows the CTC routines supported for each type of device, including Meridian-specific device types and routines.

Table D–3 Routines Supported for Meridian Devices

CTC Routine	Voice Set	Agent	Group	Route Point	Monitor Channel	Voice Channel
ctcAddMonitor					X	
ctcAnswerCall	X	X				X
ctcAssign	X	X	X	X	X	X
ctcConferenceJoin	X	X				
ctcConsultationCall	X	X				
ctcDeassign	X	X	X	X	X	X
ctcErrMsg	X	X	X	X	X	X
ctcGetChannelInformation	X	X	X	X	X	X
ctcGetEvent	X	X	X	X	X	X
ctcGetMonitor	X	X	X	X		X
ctcGetRouteQuery				X		
ctcHangupCall	X	X				
ctcHoldCall	X	X				
ctcMakeCall	X					

Table D–3 Routines Supported for Meridian Devices (Continued)

CTC Routine	Voice Set	Agent	Group	Route Point	Monitor Channel	Voice Channel
ctcReconnectHeld	X	X				
ctcRemoveMonitor					X	
ctcRespondToRouteQuery				X		
ctcRetrieveHeld	X	X				
ctcSetAgentStatus	X	X				
ctcSetCallForward	X					
ctcSetDoNotDisturb	X					
ctcSetMessageWaiting	X					
ctcSetMonitor	X	X	X	X		X
ctcSingleStepTransfer	X	X				X
ctcTransferCall	X	X				
ctcWinGetEvent	X	X	X	X		X
ctcWinGetRouteQuery				X		
Meridian-Specific Routines						
ctcMlpCloseVoiceFile						X
ctcMlpCollectDigits						X
ctcMlpLogoffMailBox						X
ctcMlpLogonMailBox						X
ctcMlpOpenVoiceFile						X
ctcMlpPlayMessage						X

Sections D.4.2 to D.4.6 provide more information about assigning to supported devices.

D.4.2 Assigning to Voice Sets

To assign to a voice set (terminal), you specify its telephone number or extension number. This is known as its DN (Directory Number). You must assign a channel to a voice set if you want to use CTC to make outgoing calls. Channels assigned to ACD agents cannot be used for outgoing calls.

Note that before you assign to a voice set, your switch administrator must configure the switch. For more information, refer to Section D.3.1.

Standard Telephones (500 and 2500 Sets)

If you are using Meridian switch software X11 Release 17 or earlier, the following routines are not supported for channels assigned to 500 or 2500 sets:

ctcConferenceJoin
ctcHangupCall
ctcTransferCall

D.4.3 Assigning to ACD Agents

You assign a channel to an ACD agent by specifying the agent's position identifier (position ID). The position ID uniquely identifies an agent in an ACD group (ACD queue), and is defined by the switch administrator (see Section D.3.1).

Note that when you assign a channel to an ACD agent:

- You cannot place outgoing calls from the agent (this is a restriction on the Meridian switch).

  However, this restriction can be avoided if the agent is using a multiline set. You can assign two channels:

  - A channel to the agent's position ID. This automatically identifies a line on the set that has been configured for the agent's use.
  - A channel to the DN of another line on the set (a line that has not been configured as an agent's position, also known as an Individual DN (IDN)).

  The agent receives incoming calls and related status information on the line associated with the position ID, and uses the channel assigned to the other line to place outgoing calls.

- If you are monitoring the agent, the Meridian switch supplies call status information on incoming calls.

D.4.4 Assigning to ACD Group Numbers

You assign a channel to an ACD group (also known as an ACD queue) by specifying the ACD DN for the call queue, as defined by the switch administrator.

D.4.5 Assigning to Route Points

If you want to use call routing, you must assign a channel to a route point. A route point is a logical device associated with a Controlled DN (CDN). A CDN is similar to an ACD queue but it has no agents. When a call enters the queue, the ctcGetRouteQuery and ctcRespondToRouteQuery routines can be used to route the call.

Note that the Meridian switch administrator must configure the switch so that the CDNs operate in controlled mode.

For channels assigned to route points, only the following events are supported:

ctcK_InboundCall
ctcK_OpDisconnected

For more information about these events, refer to the description of ctcGetEvent in Chapter 2.

D.4.6 Assigning to Voice Channels

Assigning to a voice channel enables you to use CTC routines to access voice services from Meridian Mail. These are described in Section D.16.

To assign to a voice channel, specify:

- The value ctcK_VoiceChannel in the deviceType field of the ctcAssignData structure.
- The voice channel class number (as defined by the switch administrator) in the deviceDN field of the ctcAssignData structure.
- The value ctcK_MeridianLink in the APIextensions field of the ctcAssignData structure. This enables you to access Meridian-specific functions.

For more information about Meridian Mail, refer to your Meridian documentation.

D.5 ctcCancelCall

This routine is not supported by Meridian switches. To cancel a consultation call and retrieve the original call, you must use ctcReconnectHeld.

For example:

1. You use ctcConsultationCall to place a call to another party. The original call is placed on consultation hold.
2. When there is no answer from the consultation party, you use ctcReconnectHeld. This cancels the consultation call and returns you to the original call.

D.6 ctcConsultationCall

This section describes operating differences and points to note when you use ctcConsultationCall with a Meridian switch. For a full description of this routine, refer to Chapter 2.

D.6.1 consultType Values

Meridian switches require you to specify the type of consultation call you are making. Specify one of the following with the consultType argument:

- ctcK_ConsultTransfer for a transfer call
- ctcK_ConsultConference for a conference call

ctcK_ConsultGeneric is not supported.

D.6.2 callRefId and newCallRefId

The following table shows how the Meridian supports call references for the callRefId and newCallRefId arguments:

For this argument...	A Meridian switch...
callRefId	Does not use the value you specify. However, for compatibility with other switches, Dialogic recommends you pass the reference identifer for the call as returned by ctcGetEvent, ctcWinGetEvent, or ctcMakeCall.
newCallRefId	Returns one of the following: • If the Meridian switch is running switch software X11 Release 18 or earlier, the value zero. • If the Meridian switch is running switch software X11 Release 19 or later, the call reference for the new call.

D.6.3 applicationData

Meridian switches do not support application data for a call. Pass the address of a zero-length character string with the applicationData argument.

D.6.4 ctcBadObjState Returned for Initiating a Call Transfer

If the condition value ctcBadObjState is returned when you initiate a call transfer with ctcConsultationCall, the original party (for example, the calling party) may have abandoned the call.

You must ensure that the call is cleared. Check CTC events:

- If the ctcK_Disconnected event is returned, the call has been cleared.
- If no call event is returned, use ctcHangupCall to clear the call.

  Note that you may receive ctcBadObjState in response to this routine. This indicates that the call has already been cleared but no ctcK_Disconnected event was logged. A call event may not be returned if, for example, the call is external.

D.7 ctcGetChannelInformation

This section describes operating differences and points to note when you use ctcGetChannelInformation with Meridian switches. For a full description of this routine, refer to Chapter 2.

D.7.1 Line Type Values

Meridian switches can return the following lineType values:

ctcK_LineRoutePoint
ctcK_LineVoiceSet
ctcK_LineVRU

Note that:

- ctcK_LineVoiceSet is returned for channels assigned to a telephony device, (telephones or terminals, ACD groups, ACD agents, or VRUs).
- ctcK_LineVRU is returned if the channel is assigned to a voice channel only (see Section D.4.6).
- ctcK_LineDataSet and ctcK_LineTrunk are not supported.

D.7.2 Prime Values

Meridian switches do not return information in the prime field of the ctcChanData structure.

D.7.3 Set Type Values

Meridian switches do not return information in the setType field of the ctcChanData structure.

D.7.4 Switch-Specific Support

If you are using switch-specific routines provided by CTC for Meridian switches (see Section D.16) for a channel assigned to a voice channel, the following values can be returned in the switchSpecificSupport field of the ctcChanData structure:

ctcM_MlpCloseVoiceFile
ctcM_MlpCollectDigits
ctcM_MlpLogoffMailBox
ctcM_MlpLogonMailBox
ctcM_MlpOpenVoiceFile
ctcM_MlpPlayMessage

D.8 ctcGetEvent and ctcWinGetEvent

This section describes operating differences and points to note when you use ctcGetEvent or ctcWinGetEvent with a Meridian switch. For full descriptions of these routines, refer to Chapter 2.

D.8.1 Fields Used in the ctcEventData Structure

Table D–4 shows which fields in the ctcEventData structure are supported by Meridian switches. If a field is *Not supported*, CTC always returns null data for that field.

Table D–4 Event Information Supported by Meridian Switches

Field	Support
refId	See Section D.8.2.
netCallId	As described in Chapter 2.
oldRefId	As described in Chapter 2.
oldNetCallId	As described in Chapter 2.
state	See Section D.8.3.
event	See Sections D.8.4, D.8.5, D.8.6, D.8.7, D.8.8, and D.8.9.
eventQualifier	See Section D.8.10.
type	See Section D.8.11.
otherPartyType	As described in Chapter 2.
otherPartyQualifier	Not supported.
otherParty	See Sections D.8.12 and D.8.17.
otherPartyTrunk	As described in Chapter 2.
otherPartyGroup	As described in Chapter 2.

Table D–4 Event Information Supported by Meridian Switches (Continued)

Field	Support
thirdPartyType	As described in Chapter 2.
thirdPartyQualifier	Not supported.
thirdParty	See Sections D.8.12 and D.8.17.
thirdPartyTrunk	As described in Chapter 2.
thirdPartyGroup	As described in Chapter 2.
calledPartyType	As described in Chapter 2.
calledPartyQualifier	Not supported.
calledParty	See Sections D.8.12 and D.8.17.
calledPartyTrunk	As described in Chapter 2.
calledPartyGroup	As described in Chapter 2.
applicationData	Not supported.
monitorParty	As described in Chapter 2.
nestedMonitorChannel	As described in Chapter 2.
agentMode	See Section D.8.13.
agentId	Not supported.
agentGroup	As described in Chapter 2.
agentData	Not supported.
logicalAgent	Not supported.
dtmfDigits	See Section D.8.14.
originatingPartyType	See Section D.8.15.
originatingPartyQualifier	Not supported.
originatingParty	See Section D.8.15.
originatingPartyTrunk	Not supported.
originatingPartyGroup	Not supported.
secOldRefId	Not supported.
callsQueued	Not supported.
accountInfo	Not supported.

Table D–4 Event Information Supported by Meridian Switches (Continued)

Field	Support
timeStamp	See Section D.8.16.
privateData	Not supported.

D.8.2 Call Reference Identifiers Returned for Events

If you are running Meridian switch software X11 Release 16 through X11 Release 18, the reference identifier for a call may not be the same for all events, that is, it cannot be predicted accurately.

D.8.3 Call States

The state transition for an incoming call is as follows:

Null → **Receive** → **Initiate** → **Active** → **Null**

Note that in this transition, the call is answered after the Receive state. When it is answered, the state of the call is Initiate followed by Active, and when the call is hung up, the state is Null.

D.8.4 Group Events

If you are monitoring a channel assigned to an ACD queue, the only event returned is ctcK_OpDisconnected (the caller has hung up before the call was answered).

D.8.5 Route Point Events

If you are monitoring a channel assigned to a route point, the only events returned are:

ctcK_InboundCall (Queued state)
ctcK_OpDisconnected

D.8.6 Agent Events

The Meridian supports the ctcK_AgentModeChange event.

For switches supporting X11 Release 22 or later and Meridian Link Release 5.0 or later, the following additional agent events are supported:

ctcK_AgentLoggedOn
ctcK_AgentLoggedOff

D.8.7 Call Events Not Supported

The following call events are not supported by Meridian switches:

ctcK_DestNotObtainable
ctcK_Diverted
ctcK_OffhookPrompt
ctcK_OpRetrieved
ctcK_Unavailable

The following call events are supported only if the Meridian switch is running X11 Release 22 or later:

ctcK_OpHeld
ctcK_OpConferenced
ctcK_TpSuspended
ctcK_Transferred

D.8.8 Switch-Specific Call Events

If you are using switch-specific routines provided by CTC for Meridian switches (see Section D.16), the following additional call events can be returned:

ctcK_MlpDigitsCollected
ctcK_MlpEndOfPlay

See the descriptions of the ctcMlpCollectDigits routine and ctcMlpPlayMessage routine for more information.

D.8.9 Call Events and States

Table D–5 shows the device states that can be returned for each **call** event.

Table D–5 Call Events and States Returned

Event	Description	States
ctcK_DestBusy	The dialed destination is busy.	Fail
ctcK_DestChanged	The call from the assigned device was redirected to another destination.	Deliver
ctcK_DestInvalid	The attempted call has failed.	Fail
ctcK_DestSeized	A call has been successfully dialed. If this call is external to the ACD, the network number has been verified and the outbound trunk seized. This does not indicate that the other end is actually ringing or answered.	Deliver

Table D–5 Call Events and States Returned (Continued)

Event	Description	States
ctcK_Error	The call has failed for an unspecified reason.	Fail
ctcK_InboundCall	A new call has arrived at the assigned device prior to routing.	Receive or Queued
ctcK_Offhook	A new call has been made from the assigned device.	Initiate
ctcK_OpAnswered	The other party has answered the call from the assigned device.	Active or Queued
ctcK_OpDisconnected	Either the other party hung up before the call was answered, or, for switches supporting Meridian switch software X11 Release 22 or later and Meridian Link software Release 5.0 or later, the other party in a conference call hung up.	Active Null
ctcK_Other	An event has occurred during the call (see Section D.8.10).	Deliver or Active Unknown
ctcK_TpAnswered	A call has been connected to this party.	Active
ctcK_TpConferenced	This party has been connected in a conference call.	Active
ctcK_TpDisconnected	A call has been disconnected from this party possibly because it has been transferred.	Null
ctcK_TpRetrieved	The held call has been retrieved by this party.	Active

D.8.10 Call Event Qualifiers

This section describes Meridian switch qualifiers for call events. Call events occur during the progress of a call and, along with call states, indicate the success or failure of calls on the monitored device. The qualifier can sometimes provide more information on the nature of the event.

CTC returns information about call events in the ctcEventData structure returned by the ctcGetEvent or ctcWinGetEvent routine. Meridian switches supply more detailed information on events, and CTC returns this additional information in the eventQualifier field of the structure.

To determine which event qualifier has been returned, compare the value in the eventQualifier field with the literals listed as ctcK_Mlp... in Table D–6. These literals define the possible qualifiers returned by a Meridian switch and are supplied in a CTC definitions file installed on your system (see Section 1.5).

Table D–6 Call Event Qualifiers for Meridian Switches

Qualifier Name	Description
ctcK_MlpAcdQueued	Call is queued
ctcK_MlpAcdRinging	ACD queue found and is ringing
ctcK_MlpAttendQueued	Call is queued
ctcK_MlpCallAbandon†	Calling party has disconnected
ctcK_MlpCallForward	Incoming call is forwarded from another destination
ctcK_MlpCallForwardBusy	Incoming call is forwarded because the original destination is busy
ctcK_MlpCallForwardDnD†	Incoming call is forwarded because the original destination has set Do-Not-Disturb
ctcK_MlpCallForwardNoAnswer	Incoming call is forwarded because the original destination did not answer
ctcK_MlpCallPickup†	Incoming call is picked up but the other party has disconnected
ctcK_MlpConfComplete	Conference joined
ctcK_MlpConAck	CON message acknowledged
ctcK_MlpCOSNotConfig	Transfer Class Of Service (COS) not configured
ctcK_MlpDigitCollectSuccess	DTMF digits have been collected successfully
ctcK_MlpDirect	Direct incoming call
ctcK_MlpFastTransferDone	Fast transfer completed successfully
ctcK_MlpInterDigitTimeout	Inter digit timeout received
ctcK_MlpInvCustNumber	Invalid customer number
ctcK_MlpInvDTMFString	Invalid DTMF string received

†Supported by Meridian Link software Release 4.0 or later.

Table D–6 Call Event Qualifiers for Meridian Switches (Continued)

Qualifier Name	Description
ctcK_MlpInvOpDn	Invalid called DN
ctcK_MlpInvOpManner	Invalid terminating manner
ctcK_MlpInvOpTn	Invalid called TN
ctcK_MlpInvTpDn	Invalid calling DN
ctcK_MlpInvTpManner	Invalid originating manner
ctcK_MlpInvTpTn	Invalid calling TN
ctcK_MlpInvTpUserType	Invalid calling user type
ctcK_MlpKeyBufferOverflow	Key buffer overflow occurred
ctcK_MlpMultAppearanceDn	DN appears on more than one set
ctcK_MlpOffNightService†	The attendant goes off night service while the night service DN is ringing
ctcK_MlpOpAnswered	Called party has answered
ctcK_MlpOpBadState	Called party is in a bad state
ctcK_MlpOpBlocking	Called party is blocked
ctcK_MlpOpBusy	Called party is busy
ctcK_MlpOpRinging	Called party is ringing
ctcK_MlpOpTransferredCall	Called party transferred call
ctcK_MlpQueued	Call is queued
ctcK_MlpReadyState	Calling party's phone is ready
ctcK_MlpRetrieveComplete	Return to original call complete
ctcK_MlpSetInConfCall	Set active in conference call
ctcK_MlpSignalling	Calling party is receiving end-to-end signaling
ctcK_MlpSystemError	System error
ctcK_MlpTpAccessRestrict	Access restriction
ctcK_MlpTpBlocking	Calling party is blocking
ctcK_MlpTpBusy	Calling party is busy
ctcK_MlpTpDisconnect	Calling party is disconnected

†Supported by Meridian Link software Release 4.0 or later.

Table D–6 Call Event Qualifiers for Meridian Switches (Continued)

Qualifier Name	Description
ctcK_MlpTpInuse	Calling party DN is in use
ctcK_MlpTpInvokedHold	Calling party invoked hold
ctcK_MlpTpMaintenance	Calling party set maintenance is busy
ctcK_MlpTpOnhook	Calling party is on-hook
ctcK_MlpTpPermHold	Calling party is on permanent hold
ctcK_MlpTpRinging	Incoming call on calling party set
ctcK_MlpTpUnableToAnswer	Cannot answer incoming call
ctcK_MlpTpUnableToDisconnect	Cannot disconnect calling party
ctcK_MlpTpUnableToPutOnHold	Cannot put call on hold
ctcK_MlpTransferKeyNotConfig	Transfer key not configured
ctcK_MlpTransferKeyNotIdle	Transfer key not idle
ctcK_MlpTrunkSeized	Calling trunk seized
ctcK_MlpUnableCompConf	Unable to complete conference
ctcK_MlpUnableCompFT	Unable to complete fast transfer
ctcK_MlpUnableCompRetr	Unable to complete retrieve
ctcK_MlpUnableCompXfer	Unable to complete transfer
ctcK_MlpUnableInitFT	Unable to initiate fast transfer
ctcK_MlpUnableInitXfer	Unable to initiate transfer
ctcK_MlpUnknown	Unknown event
ctcK_MlpXferComplete	Call successfully transferred

D.8.11 Call Types

The Meridian supports the following call types to an ACD agent:

For an incoming call...	For an incoming call that rings off...
ctcK_MlpCallForward ctcK_MlpCallForwardBusy ctcK_MlpCallForwardDnD ctcK_MlpDirect	ctcK_MlpCallAbandon ctcK_MlpCallForwardNoAnswer ctcK_MlpCallPickup ctcK_MlpOffNightService

D.8.12 Other, Third, and Called Party Information

For a Meridian switch, two separate party items can be returned in the same field of the ctcEventData structure. These items are separated by the character /.

For example, if you call an ACD agent from the assigned device and the agent answers (ctcK_OpAnswered), the otherParty field of the ctcEventData structure can contain both the DN for the ACD queue and a position ID for the ACD agent. If 3776 is the DN for the ACD queue and 7892 is the position ID for the ACD agent on that queue, the otherParty field will contain 3776/7892. The / character is used to separate these items.

Other, Called, and Third Parties

CTC supports the Calling Line ID (CLID) feature supplied by Meridian switches. CLID information is returned in the otherParty and thirdParty fields of the ctcEventData structure.

A Meridian switch passes the information over the CTC link only if:

1. The DNIS software package and the appropriate ISDN cards are installed on the switch.
2. There is an ISDN trunk connecting the switch to the Central Office.
3. The switch is configured correctly (see the switch administrator).

D.8.13 Agent Modes

This section provides information about data returned in the agentMode field.

ctcK_AgentInService

The Meridian supports an additional value, ctcK_AgentInService, that can be returned in the agentMode field. This value is returned when an agent becomes available after being unable to receive calls because of being involved in other work (agent mode ctcK_AgentOtherWork).

Switch Software Release

The agent mode values supported by a Meridian are dependent on the release of Meridian switch software running on the switch:

Value	Release of Meridian Switch Software Required
ctcK_AgentInService	X11 Release 19 or later
ctcK_AgentOtherWork	X11 Release 19 or later

Value	Release of Meridian Switch Software Required
ctcK_AgentLogin	X11 Release 22 or later
ctcK_AgentLogout	X11 Release 22 or later
ctcK_AgentReady	X11 Release 22 or later
ctcK_AgentNotReady	X11 Release 22 or later

Work Modes for Devices

The value returned in the agentMode field may not always reflect the work mode for the agent. Instead, it can return the mode for the device (for example, telephone set) that the agent is using. For example, ctcK_AgentInService may be returned when the agent has logged out. This is because the DN associated with the agent's telephone set is still in service.

D.8.14 DTMF Digits

Meridian switches return information in the dtmfDigits field only if the application uses CTC API extensions for the Meridian (see Section D.16) and the channel is assigned to a voice channel.

Any DTMF digits entered by the other party are returned in this field. For more information, refer to the description of the ctcMlpCollectDigits routine in this appendix.

D.8.15 Originating Party Information

Meridian switches can use the following Originating Party fields:

Field	Description
originatingPartyType	The Meridian supports the value ctcK_Dn only.
originatingParty	This field contains a treatment number, if available. The maximum length for the treatment number is specified by the literal ctcMaxDnLen in a CTC definitions file (see Section 1.5). Note that this maximum length includes the null termination character (NUL).

D.8.16 Time Stamp

If the link is configured to return a time stamp from the switch and not the CTC server, the switch can return information. However, the following conditions apply:

- The Meridian switch must be running Meridian switch software X11 Release

22 or later and Meridian Link software release 5.0 or later.

- The Meridian switch can provide information in the following ctcTimeStamp fields only:

  hour
  minute
  second

- Time stamp information may not be provided for all events.

  To ensure that time stamp information is returned for each CTC event, Dialogic recommends that you configure the link to return time stamp information generated by the CTC server. For more information, refer to the *CT-Connect Installation and Administration Guide*.

D.8.17 Party Information and Events

Table D–7 shows the party information specific to Meridian call events.

Table D–7 Meridian Party Information for Call Events

Event	Field	Party Information	Explanation
ctcK_DestBusy			
	Other	Busy Party or Called Number†	
	Third	Null	
	Called	Null	
ctcK_DestChanged			
	Other	Ringing Party or Called Number†	
	Third	Null	
	Called	Null	
ctcK_DestInvalid			
	Other	Invalid Destination Number or Called Number†	
	Third	Null	
	Called	Null	

†If a call is forwarded to a nonagent, the called number is provided in the other party field.

Table D–7 Meridian Party Information for Call Events (Continued)

Event	Field	Party Information	Explanation
ctcK_DestSeized			
	Other	Ringing Party or Called Number†	
	Third	Null	
	Called	Null	
ctcK_Error			
	Other	Null	
	Third	Null	
	Called	Null	
ctcK_InboundCall			
	Other	Calling Party	
	Third	Transferred or Last Redirected ACD DN if available	If the channel is assigned to an ACD agent and the call was forwarded, the last station called before redirection.
	Called	Called Number if available or DNIS if available	
ctcK_Offhook			
	Other	Null	
	Third	Null	
	Called	Null	
ctcK_OpAnswered			
	Other	Answering Party	
	Third	Null	
	Called	Null	

†If a call is forwarded to a nonagent, the called number is provided in the other party field.

Table D–7 Meridian Party Information for Call Events (Continued)

Event	Field	Party Information	Explanation
ctcK_OpDisconnected			
	Other	Calling Party	For a conference call, the remaining party in the call.
	Third	Null	
	Called	Called Number	
ctcK_Other			
	Other	Null	
	Third	Null	
	Called	Null	
ctcK_TpAnswered			
	Other	Party at Other End of Call	
	Third	Null	
	Called	Null	
ctcK_TpConferenced			
	Other	Null	
	Third	Null	
	Called	Null	
ctcK_TpDisconnected			
	Other	Null	
	Third	Null	
	Called	Null	
ctcK_TpRetrieved			
	Other	Retrieved party	
	Third	Null	
	Called	Null	

This table gives only Meridian party information. For other switch-specific information, refer to the appendix for your switch (for example, Appendix C for the DEFINITY). For a description of each event (such as ctcK_DestBusy), see

Table 2–5.

D.9 ctcGetRouteQuery and ctcWinGetRouteQuery

This section describes operating differences and points to note when you use ctcGetRouteQuery or ctcWinGetRouteQuery with Meridian switches. For full descriptions of these routines, refer to Chapter 2.

D.9.1 Fields Used in the ctcRouteData Structure

Table D–8 shows which fields in the ctcRouteData structure are used by Meridian switches. If Table D–8 shows that a field is *Not supported*, CTC always returns null data for that field.

Table D–8 Route Information Supported by Meridian Switches

Field	Support
routeId	As described in Chapter 2
refId	As described in Chapter 2
spare001	Not supported
otherPartyType	As described in Chapter 2
otherParty	As described in Chapter 2
otherPartyTrunk	As described in Chapter 2
otherPartyGroup	As described in Chapter 2
thirdPartyType	Not supported
thirdParty	Not supported
thirdPartyTrunk	Not supported
thirdPartyGroup	Not supported
calledPartyType	Not supported
calledParty	As described in Chapter 2
calledPartyTrunk	Not supported
calledPartyGroup	As described in Chapter 2
applicationData	Not supported
dtmfDigits	Not supported
timeStamp	As described in Section D.9.2

Table D–8 Route Information Supported by Meridian Switches (Continued)

Field	Support
privateData	Not supported

D.9.2 Time Stamp

If the link is configured to return a time stamp from the CTC server, the timeStamp field contains the data and time the CTC server received the route request. For more information, refer to the description of this field in Chapter 2.

Meridian switches do not support time stamp information for channels assigned to route points. If the link is configured to return a time stamp from a Meridian switch, this field contains null data.

For more information about changing the configuration of a link, refer to the *CT-Connect Installation and Administration Guide* for your CTC server platform.

D.10 ctcMakeCall

This section describes operating differences and points to note when you use ctcMakeCall with Meridian switches. For a full description of this routine, refer to Chapter 2.

D.10.1 Application Data

Meridian switches do not support application data for a call. Pass the address of a zero-length character string with the applicationData argument.

D.10.2 Call Reference Identifier

If a Meridian switch is running X11 Release 18 or earlier of the Meridian switch software, the value zero is returned as the reference identifier for the call. For more information, refer to the description of the ctcMakeCall callRefId argument in Chapter 2.

D.11 ctcRespondToRouteQuery

This section describes operating differences and points to note when you use ctcRespondToRouteQuery with Meridian switches. For a full description of this routine, refer to Chapter 2.

D.11.1 Responding to Route Queries

Meridian switches require that your application responds to a call within four seconds either by providing a new route or by providing a *treatment*, such as music (see Section D.11.2).

If you do not respond within that time, the Meridian provides the default call treatment (as defined on the Meridian switch).

Note that if the Meridian provides default call treatment for 10 or more calls, it no longer allows your application control of the assigned route point. To maintain control of the assigned route point, do not specify the address of a zero-length character string with the newCalledNumber argument more than 10 times.

D.11.2 Delayed Routing

If your application does not want to reroute the call immediately, or does not know where to route the call at that time, it can continue to control the call by providing music, ringback, or silence.

Use the newCalledNumber argument to specify the address of a null-terminated character string that contains one of the following:

- ##M#*nn* for music, where *nn* is a route number (in hexadecimal) that identifies the music source (see your Meridian administrator for more information)
- ##R for ringback
- ##S for silence

Restriction

Meridian switches do not accept silence (##S) as the first treatment. Use ctcRespondToRouteQuery to delay routing with music or ringing, then call ctcRespondToRouteQuery again to continue the delay with silence.

D.11.3 Application Data

Meridian switches does not support application data for a call. Pass the address of a zero-length character string with the applicationData argument.

D.12 ctcSetAgentStatus

This section describes operating differences and points to note when you use ctcSetAgentStatus with Meridian switches. For a full description of this routine, refer to Chapter 2.

D.12.1 agentMode

The Meridian supports the following agentMode values:

This value...	Specifies that...
ctcK_AgentLogin	The agent is logging in.
ctcK_AgentLogout	The agent is logging out.
ctcK_AgentReady	The agent is ready to receive calls.
ctcK_AgentNotReady	If the agent is engaged in a call, the Meridian disconnects the call and no further calls are presented to the agent (agent state is not ready).
ctcK_Mlp_AgentNotReady_ACD	If the agent is engaged in an ACD call, no further calls are presented to the agent (agent state is not ready) but the Meridian does not disconnect the call.
ctcK_Mlp_AgentNotReady_IDN	If the agent is engaged in an Internal DN (IDN) call, no further calls are presented to the agent (agent state is not ready) but the Meridian does not disconnect the call.
ctcK_Mlp_AgentNotReady_Both	If the agent is engaged in a call, no further calls are present to the agent (agent state is not ready) but the Meridian does not disconnect the call.

If you specify an unsupported agentMode, CTC returns a ctcOptNotSup error.

D.12.2 agentData and logicalAgent

When you log in an agent (ctcK_AgentLogin), the data you specify with agentData and logicalAgent is dependent on the configuration of the Meridian switch.

If the Meridian administrator has configured agent identifiers (IDs) on the switch:

For this argument...	Specify...
agentData	The agent ID. This is an identifier of up to 4 digits defined by the switch administrator. The identifier must be unique on the switch.
logicalAgent	The address of a zero-length character string.

If agent IDs are not configured on the Meridian:

For this argument...	Specify...
agentData	The address of a zero-length character string.
logicalAgent	The address of a zero-length character string.

D.13 ctcSetCallForward

This section describes operating differences and points to note when you use ctcSetCallForward for Meridian switches. For a full description of this routine, refer to Chapter 2.

D.13.1 forwardMode

The only value that the Meridian supports for the forwardMode argument is ctcK_CfAll.

D.14 ctcSingleStepTransfer

This section describes the operating differences and points to note when you use ctcSingleStepTransfer with Meridian switches. For a full description of this routine, refer to Chapter 2.

D.14.1 Switch Software Required

ctcSingleStepTransfer is supported on Meridian switches running:

- X11 Release 21 or later of the Meridian switch software
- Release 5.0 or later of the Meridian Link software

D.14.2 callRefId

Meridian switches do not use the call reference identifier specified with the callRefId argument. However, to ensure that your application is compatible with other switches, Dialogic recommends that you pass the reference identifier for the call as returned by ctcGetEvent, ctcWinGetEvent, or ctcMakeCall.

D.14.3 newCallRefId

For the newCallRefId argument, Meridian switches return the call reference for the original call.

D.14.4 Supported Devices

ctcSingleStepTransfer is supported for channels assigned to:

- DNs (for example, voice sets, ACD agents, or ACD groups)
- Voice channels

It is not supported for channels assigned to route points or monitor channels.

D.14.5 applicationData

Meridian switches do not support application data for a call. Pass the address of a zero-length character string with the applicationData argument.

D.15 ctcTransferCall

This section describes operating differences and points to note when you use ctcTransferCall with Meridian switches. For a full description of this routine, refer to Chapter 2.

D.15.1 500 and 2500 Sets

If you are using Meridian switch software X11 Release 17 or earlier, this routine is not supported for channels assigned to 500 or 2500 sets.

D.15.2 activeCallRefId

Meridian switches do not use the call reference identifier specified with the activeCallRefId argument.

However, to ensure that your application is compatible with other switches, Dialogic recommends that you pass the reference identifier for the call as returned by ctcGetEvent, ctcWinGetEvent, or ctcMakeCall.

D.15.3 heldCallRefId

Meridian switches do not use the call reference identifier specified with the heldCallRefId argument.

However, to ensure that your application is compatible with other switches, Dialogic recommends that you pass the reference identifier for the call as returned by ctcGetEvent, ctcWinGetEvent, or ctcMakeCall.

D.15.4 newCallRefId

Meridian switches do not return a new call reference for the transferred call:

- If the Meridian switch is running switch software X11 Release 18 or earlier, it returns the value zero as the newCallRefId.

- If you are using Meridian switch software X11 Release 19 or later and Meridian Link software Release 4.0 or later, it returns the latest reference identifier for the held call as returned by ctcGetEvent, ctcWinGetEvent, or ctcMakeCall.

D.15.5 ctcBadObjState Returned for Call Transfer

If you complete a call transfer with ctcTransferCall and the condition value ctcBadObjState is returned, either the calling party or the destination party has abandoned the call. Use ctcReconnectHeld to reconnect to the remaining party:

- If the calling party is connected, the destination party has hung up. You can either use ctcConsultationCall to try to call the destination party again, or abandon the consultation call.
- If the condition value ctcBadObjState is returned for the call to ctcReconnectHeld, the calling party has hung up. Use ctcHangupCall to clear the call.

D.16 CTC Routines for Meridian Switches

The following pages describe Meridian-specific CTC routines that are provided as an extension to the standard CTC API. These routines are for CTC applications that will only be used with a Meridian switch.

To use these Meridian-specific CTC routines:

- Your Meridian switch must be configured to support both CTC and Meridian Mail. The Meridian switch requires the following:
  - Meridian switch software X11 Release 19 or later.
  - Meridian Link software Release 3.0 or later with Host-Enhanced Voice Processing (service 101).
- The channel must be assigned to a voice channel (see Section D.4.6).
- When you use ctcAssign to assign the channel, you must specify the value ctcK_MeridianLink in the APIextensions field of the ctcAssignData structure. If you do not specify this value, you will not be able to use these routines for the assigned voice channel. Refer to the description of ctcAssign in Chapter 2 for more information.

ctcMlpCloseVoiceFile
Close a Voice File

Format in C

unsigned int ***ctcMlpCloseVoiceFile*** *(ctcChanId channel,*
unsigned int fileId)

Description

This routine closes an open voice file on the Meridian Mail system. Use this routine when the call on the voice channel has ended. That is, when the caller has hung up or been transferred to an agent.

For details of how to open a voice file on the Meridian Mail system, refer to the description of the ctcMlpOpenVoiceFile routine in this appendix.

Arguments

channel

type:	**ctcChanId**
access:	**read only**
mechanism:	**by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the voice channel in use.

The ctcChanId datatype is defined in a CTC definitions file (see Section 1.5).

fileId

type:	**integer (unsigned)**
access:	**read only**
mechanism:	**by value**

This 32-bit integer contains the file identifier for the file you are closing. Specify the identifier returned by the ctcMlpOpenVoiceFile routine for the file you want to close.

ctcMlpCollectDigits
Collect DTMF Digits

Format in C

*unsigned int **ctcMlpCollectDigits** (ctcChanId channel,*
unsigned int numberOfDigits,
unsigned int clearMode,
unsigned int interDigTimeout,
ctcMlpTermKeys termKeys)

Description

This routine enables you to collect DTMF digits entered by a caller. These digits are entered when the caller presses keys on a touch-tone phone, for example, to respond to a message played by ctcMlpPlayMessage.

You must call ctcGetEvent (or ctcWinGetEvent on Windows 3.1/3.11 systems) before you use this routine. For example, call:

1. ctcGetEvent to return events for the voice channel
2. ctcMlpCollectDigits to instruct the Meridian switch to send any DTMF digits that a caller enters

Note that you use ctcMlpCollectDigits **before** a caller enters the digits. When they have finished entering digits, a ctcK_MlpDigitsCollected event is returned and the dtmfDigits field of the ctcEventData structure contains the collected digits.

Your application can associate the DTMF digits with specific data, for example, account data or the DN for a destination agent or group. To transfer the caller, use the ctcSingleStepTransfer routine.

Arguments

channel

type:	**ctcChanId**
access:	**read only**
mechanism:	**by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the voice channel in use.

The ctcChanId datatype is defined in a CTC definitions file (see Section 1.5).

numberOfDigits

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer specifies the number of digits to be collected.

This can be used to check the caller's response to the Meridian Mail message. For example, if the specified number of digits are not collected, your application can play another message that prompts the caller to enter further digits.

If you do not want to set the number of expected digits, pass the value zero with this argument.

clearMode

type: **integer unsigned**
access: **read only**
mechanism: **by value**

This 32-bit integer indicates whether the key buffer is cleared. The key buffer contains the DTMF digits entered by the caller.

Specify one of the following values:

Value	Description
ctcK_MlpClearOn	Specifies that the key buffer is cleared.
ctcK_MlpClearOff	Specifies that the key buffer is not cleared.

interDigTimeout

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer specifies the time (in seconds) between each digit entered.

If the caller does not enter a digit within the specified time, CTC returns a ctcK_MlpDigitsCollected event (state failed) with the qualifier, ctcK_MlpTimeout.

If you do not want to set the time between digits, pass the value zero with this argument.

termKeys

type: **ctcMlpTermKeys**
access: **read only**
mechanism: **by reference**

This argument is the address of a character string that contains one or more digits used to indicate the end of dialing. The caller presses this key (for example, the key) when they have finished dialing.

The ASCII string can contain any combination of numbers 0 through 9 and the characters * and #. The maximum length for termKeys is specified by the literal ctcMlpMaxKeysLen in a CTC definitions file. Note that this maximum length includes the null termination character (NUL).

If you do not want to specify a termination key, pass the address of a zero-length character string.

ctcMlpLogoffMailBox
Log Off a Meridian Mail Account

Format in C

*unsigned int **ctcMlpLogoffMailBox** (ctcChanId channel)*

Description

Use this routine when you no longer need to access Meridian Mail features for the assigned voice channel. This routine logs off the voice channel from the Meridian Mail system.

For more information about logging on to a Meridian Mail system, see the description of the ctcMlpLogonMailBox routine in this appendix.

Arguments

channel

type:	**ctcChanId**
access:	**read only**
mechanism:	**by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the voice channel in use.

ctcMlpLogonMailBox
Log On to a Meridian Mail Account

Format in C

*unsigned int **ctcMlpLogonMailBox** (ctcChanId channel,*
ctcMlpUserId userId,
ctcMlpPassword password)

Description

This routine enables you to log on to a Meridian Mail account and access Meridian Mail features from the assigned voice channel. For example, if the switch receives an incoming call for the voice channel, your application can respond by playing a voice file on the Meridian Mail system. A voice file can present the caller with a menu of options and prompt them to respond by pressing telephone keys.

When you use ctcMlpLogonMailBox to log on to a Meridian Mail account, the voice channel is registered as the Meridian Mail account user.

Before you use Meridian Mail features, you must use the following sequence of routines:

1. ctcAssign to assign a channel to a voice channel
2. ctcSetMonitor to set monitoring on for the voice channel
3. ctcGetEvent so that events are returned for the voice channel
4. ctcMlpLogonMailBox to log on to a Meridian Mail account

When you have finished using Meridian Mail voice services, use ctcMlpLogoffMailBox.

Arguments

channel

type:	**ctcChanId**
access:	**read only**
mechanism:	**by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the voice channel in use.

userId

type: **ctcMlpUserId**
access: **read only**
mechanism: **by value**

This character string contains the DN for the Meridian Mail system as defined on the switch.

The maximum length for userId is specified by the literal ctcMlpUserIdLen in a CTC definitions file. Note that this maximum length includes the null termination character (NUL).

password

type: **ctcMlpPassword**
access: **read only**
mechanism: **by value**

This character string contains the password for access to the Meridian Mail system.

The maximum length for password is specified by the literal ctcMlpPasswordLen in a CTC definitions file. Note that this maximum length includes the null termination character (NUL).

ctcMlpOpenVoiceFile
Open a Voice File

Format in C

unsigned int ***ctcMlpOpenVoiceFile*** *(ctcChanId channel,*
ctcMlpFileName fileName,
*unsigned int *fileId)*

Description

This routine opens a voice file on the Meridian Mail system. A voice file contains one or more message segments that can be played to callers.

You must always use this routine before ctcMlpPlayMessage so that you can identify which voice file is played. For example, your application might use the following sequence of routines:

1. ctcMlpOpenVoiceFile to open a file that contains voice segments used for a response message.
2. ctcMlpPlayMessage to play the message.
3. ctcMlpCollectDigits to collect input from the caller. For example, the caller presses telephone keys (sends DTMF digits) to enter an account number. When the caller has finished, the ctcK_MlpDigitsCollected event is returned and the dtmfDigits field of the ctcEventData structure contains the collected digits.

   Your application associates the DTMF digits returned in the dtmfDigits field with the DN for a destination agent or group. You can then specify this DN with the calledNumber argument for ctcSingleStepTransfer.
4. ctcSingleStepTransfer to transfer the caller to the appropriate agent or group.
5. ctcMlpCloseVoiceFile to close the file.

When the call has ended, you must always use ctcMlpCloseVoiceFile to close the voice file.

Note that CTC enables you to play voice files only. You cannot use CTC to write to a voice file.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the voice channel.

fileName

type: **ctcMlpFileName**
access: **read only**
mechanism: **by value**

This character string contains the name of the voice file on the Meridian Mail system.

The ASCII string can contain any combination of numbers 0 through 9 and the characters * and #. The maximum length for fileName is specified by the literal ctcMlpFileNameLen in a CTC definitions file. Note that this maximum length includes the null termination character (NUL).

fileId

type: **integer (unsigned)**
access: **write only**
mechanism: **by reference**

This argument is the address of a 32-bit integer that receives a file identifier for the opened voice file. You use the returned file identifier with the ctcMlpPlayMessage routine.

ctcMlpPlayMessage
Play a Voice Message

Format in C

```
unsigned int ctcMlpPlayMessage (ctcChanId        channel,
                                unsigned int     fileId,
                                unsigned int     clearMode,
                                unsigned int     interruptMode,
                                unsigned int     numberOfSegments,
                                unsigned short   *fileSegments)
```

Description

This routine plays a message to a caller. The voice file containing the message must be opened with ctcMlpOpenVoiceFile before it can be played.

The message can consist of one or more voice segments in the open voice file. Using ctcMlpPlayMessage, you can specify:

- The number of voice segments that will be played
- Which voice segments are played to make up the message
- Whether the user can interrupt the message by pressing a key on their telephone

When the message has been played (or interrupted), the ctcK_MlpEndOfPlay event is returned.

After ctcMlpPlayMessage, you can use other routines, for example, the ctcMlpCollectDigits routine to collect DTMF digits entered by the caller in response to the message. However, at the end of the call, you must use ctcMlpCloseVoiceFile to close the voice file.

Arguments

channel

type: **ctcChanId**
access: **read only**
mechanism: **by value**

This argument is a ctcChanId datatype that contains the channel identifier (channel ID) value returned by ctcAssign for the voice channel.

fileId

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer specifies the file identifier for the opened voice file. Specify the file identifier returned by the ctcMlpOpenVoiceFile routine.

clearMode

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer indicates whether the key buffer is cleared before playing commences. The key buffer contains the DTMF digits entered by the caller.

Specify one of the following values:

Value	Description
ctcK_MlpClearOn	Specifies that the key buffer is cleared before playing commences.
ctcK_MlpClearOff	Specifies that the key buffer is not cleared.

Note that if you clear the buffer and the caller presses a key to interrupt the message, the buffer will contain the interrupt key when playing has finished.

interruptMode

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer indicates whether the played message can be interrupted by pressing a key.

Specify one of the following values:

Value	Description
ctcK_MlpInterruptOn	Specifies that the caller can interrupt the message by pressing a key.
ctcK_MlpInterruptOff	Specifies that the caller cannot interrupt the message by pressing a key.

ctcMlpPlayMessage

numberOfSegments

type: **integer (unsigned)**
access: **read only**
mechanism: **by value**

This 32-bit integer specifies the number of voice segments that will be played from the open voice file. This value is used to determine the number of offsets that are referenced by the fileSegments argument.

fileSegments

type: **short (unsigned)**
access: **read only**
mechanism: **by reference**

This argument is the address of an array of words. Each word contains a file segment offset. This is a hex number between 1 and 1000 that points to the start of a voice segment in the open file. By specifying the offset for the segment, you can determine which voice segments are played.

The number of words in the array is defined by the value you specify with the numberOfSegments argument.

The maximum number of words in the array is specified by the literal ctcMlpMaxSegLen in a CTC definitions file (see Section 1.5).

Index

A

B

C

D

E

I

L

M

N

O

P

Q

R

S

T

U

V

W