

CT-Connect for CompuCALL

Programming Guide

Order Number: 05-0807-002

Revision/Update Information: This manual is an update

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

Copyright © Dialogic Corporation 1998. All Rights Reserved.

All contents of this document are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation. Every effort is made to ensure the accuracy of this information. However, due to ongoing product improvements and revisions, Dialogic Corporation cannot guarantee the accuracy of this material, nor can it accept responsibility for errors or omissions. No warranties of any nature are extended by the information contained in these copyrighted materials. Use or implementation of any one of the concepts, applications, or ideas described in this document or on Web pages maintained by Dialogic, may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not condone or encourage such infringement. Dialogic makes no warranty with respect to such infringement, nor does Dialogic waive any of its own intellectual property rights which may cover systems implementing one or more of the ideas contained herein. Procurement of appropriate intellectual property rights and licenses is solely the responsibility of the system implementer. The software referred to in this document is provided under a Software License Agreement. Refer to the Software License Agreement for complete details governing the use of the software.

DIALOGIC is a registered trademark and CT-Connect is a trademark of Dialogic Corporation.

CompuCALL, DMS-100, Meridian, Nortel, and SL-100 are trademarks of Northern Telecom.

DECnet, Digital, OpenVMS, and VAX are trademarks of Digital Equipment Corporation.

HP and HP-UX are registered trademarks of Hewlett-Packard Co.

IBM and OS/2 are registered trademarks and OS/2 Warp is a trademark of International Business Machines Corporation.

Microsoft and MS-DOS are registered trademarks, and Windows and Windows NT are trademarks of Microsoft Corporation.

NetBIOS is a trademark of Micro Computer Systems, Inc.

Novell is a registered trademark of Novell, Inc.

SCO is a registered trademark, and SCO OpenServer is a trademark of The Santa Cruz Operation, Inc. in the United States and other countries.

Solaris is a trademark of Sun Microsystems, Inc. in the United States and other countries.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd.

All other trademarks and registered trademarks are the property of their respective owners.

Publication date: December 1, 1998

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

Contents

About This Manual	vii
1 Programming Overview	
1.1 Overview of CTC/CMP for CompuCALL	1-1
1.1.1 Supported Devices	1-2
1.2 Introduction to CTC/CMP Routines	1-3
1.2.1 Routines That Control the Logical Channel	1-3
1.2.2 Routines That Perform Telephony Functions	1-4
1.3 Sequence for Calling CTC/CMP API Routines	1-5
1.4 Format of Routines	1-6
1.5 Use of Arguments	1-6
1.5.1 Data Type	1-6
1.5.1.1 CTC/CMP Data Type Definitions	1-7
1.5.2 Access to Data	1-8
1.5.3 Passing Mechanism	1-8
1.5.4 Passing Optional Data	1-9
1.6 Definitions	1-9
1.7 Constants	1-10
1.8 Errors and Condition Values for Status Returns	1-10
1.8.1 Link Problems	1-10
1.9 Exception Handling	1-11
1.10 Calling CTC/CMP Routines	1-11
1.11 CTC/CMP and Multithreaded Programming	1-11
1.11.1 Threads	1-11
1.11.2 Multithreaded Programming	1-11
1.11.3 Thread Execution	1-12
1.11.4 Using Multithreaded Programming with CTC/CMP	1-12
1.11.5 Creating a Multithreaded Program	1-12
1.12 Compiling and Linking Your Program	1-13
1.12.1 Digital UNIX Client	1-13

1.12.2	HP-UX Client	1-14
1.12.3	OpenVMS Client	1-14
1.12.4	OS/2 Client	1-15
1.12.5	SCO OpenServer Client	1-15
1.12.6	Solaris Client	1-15
1.12.7	Windows NT and Windows 95 Clients	1-16
1.13	Changes to CTC/CMP for Version 3.0	1-17
1.14	Compatibility With Previous Versions of CTC/CMP	1-18

2 Routine Specifications

ctcCmpAddMonitor	2-2
ctcCmpAnswerCall	2-7
ctcCmpAssign	2-9
ctcCmpClearCall	2-17
ctcCmpConferenceJoin	2-19
ctcCmpConsultationCall	2-21
ctcCmpDeassign	2-25
ctcCmpErrMsg	2-26
ctcCmpGetACDGroupStatus	2-28
ctcCmpGetAgentStatus	2-33
ctcCmpGetChannelInformation	2-34
ctcCmpGetEvent	2-38
ctcCmpGetMessageWaiting	2-55
ctcCmpGetMonitor	2-57
ctcCmpGetRouteQuery	2-59
ctcCmpGetRoutingEnable	2-69
ctcCmpGiveTreatment	2-70
ctcCmpHangupCall	2-73
ctcCmpHoldCall	2-75
ctcCmpMakeCall	2-77
ctcCmpReconnectHeld	2-82
ctcCmpRemoveMonitor	2-84
ctcCmpRespondToRouteQuery	2-86
ctcCmpResumeCall	2-89
ctcCmpSetAgentStatus	2-91
ctcCmpSetMonitor	2-94
ctcCmpSetRoutingEnable	2-97
ctcCmpSnapshot	2-100
ctcCmpTransferCall	2-102

3 Error and Condition Values Returned

Index

Figures

1-1	Example CTC/CMP Network	1-1
-----	-------------------------------	-----

Tables

1-1	Controlling the Communications Channel	1-3
1-2	Telephony Functions	1-4
1-3	CTC/CMP Data Types	1-7
1-4	Summary of Changes to CTC/CMP for V3.0	1-17
2-1	Routines Supported by Device Type	2-10
2-2	Call States Returned by ctcCmpGetEvent	2-42
2-3	Agent Events Returned by ctcCmpGetEvent	2-43
2-4	Call Events Returned by ctcCmpGetEvent	2-43
2-5	Event Qualifiers Returned by ctcCmpGetEvent	2-44
3-1	Condition Values Returned	3-2

About This Manual

This manual describes how to use Dialogic's CT-Connect for CompuCALL (CTC/CMP) software to write telephony applications. It provides an overview of CTC/CMP and describes all CTC/CMP Application Programming Interface (API) routines.

CTC/CMP software enables a telephony application to access the features of a:

- Nortel™ DMS-100™ switch using the CompuCALL interface
- Nortel SL-100™ switch using the Meridian™ SCAI interface

For simplicity, the term DMS-100 is used throughout this manual to refer to both the DMS-100 and the SL-100 switch, and the term CompuCALL is used to refer to both the CompuCALL interface and the Meridian SCAI interface.

When this manual refers to specific versions of the CompuCALL software, the switch load NA00*n* convention is used. Refer to the following table to see how the NA00*n* number matches other version numbers used to specify the CompuCALL software:

Switch Load NA00 <i>n</i> Convention†	CompuCALL Version	Version Number Used in CTC/CMP Configuration
NA003/4	BCS36	36
NA005	SCAI07	7
NA006	SCAI08	8
NA008	SCAI10	10
NA009	SCAI11	11

† MSL, EUR, and ASP conventions may vary.

Audience

This manual is for programmers writing computer-integrated telephony applications that run on client systems and use a link between a CTC/CMP server (a system running the CTC/CMP server software) and a DMS-100.

The manual assumes that readers are familiar with:

- Writing programs in C
- Compiling and linking programs on the operating system(s) used by your CTC/CMP clients (systems running the CTC/CMP API software).

You must also have an understanding of the components in a CTC/CMP network. Chapter 1 provides an overview of CTC/CMP systems and how they are linked to the DMS-100.

Associated Documentation

CTC/CMP Documentation

In addition to this manual, the CTC/CMP documentation set contains:

- *CT-Connect for CompuCALL Installation and Administration Guide*

This manual describes how to install the CTC/CMP software on supported platforms. It also describes administration tasks and provides basic problem solving procedures.

- *CT-Connect for CompuCALL Release Notes*

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

CompuCALL Documentation

Refer to your CompuCALL documentation for details of features and any limitations that may affect the operation of the CTC/CMP software.

Terms and Definitions

The following terms are used throughout this manual:

Term	Definition
OpenVMS	Refers to the OpenVMS™ Alpha operating system.
OS/2	Refers to OS/2 Warp™.
DMS-100	Refers to Nortel's DMS-100 switch and SL-100 switch.
CompuCALL	Refers to Nortel's CompuCALL interface and Meridian SCAI interface.
CTC/CMP	Refers to CT-Connect for CompuCALL.
CTC/CMP server	A Windows NT™ personal computer running the CTC/CMP server software.
CTC/CMP client	A supported system running the CTC/CMP API software.
Communications link	The logical link between the CTC/CMP server and the DMS-100.

Conventions

The following table describes conventions used in this manual:

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

Programming Overview

This chapter provides an overview of CTC/CMP and CTC/CMP API routines.

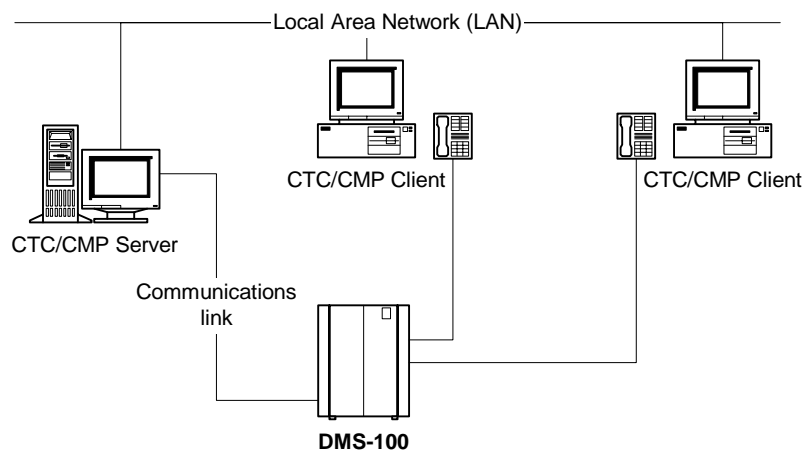
1.1 Overview of CTC/CMP for CompuCALL

CTC/CMP is a software toolkit for developing and running computer-integrated telephony applications.

A CTC/CMP application creates logical channels to telephony devices (for example, telephones, agent positions, or agent groups) so that it can perform call-processing at the devices and obtain call and party data.

The application runs on a CTC/CMP client system, which passes requests to, and receives information from, the CTC/CMP server. Figure 1-1 shows an example CTC/CMP network.

Figure 1-1 Example CTC/CMP Network



The CTC/CMP server acts as an intermediary, passing messages between CTC/CMP clients and the DMS-100.

The link between the CTC/CMP server and the DMS-100 is known as the communications link, and is identified on the CTC/CMP server by a logical identifier. This is defined when the link is created. For more information about creating and configuring communications links, refer to the *CT-Connect for CompuCALL Installation and Administration Guide*.

1.1.1 Supported Devices

You can create logical channels over a communications link to:

- Telephones or agent positions

A logical channel to one of these devices enables your application to perform basic call-processing and functions usually accessed through the telephone set or at the agent position. It also enables the application to make use of call data and party information.

- ACD groups

A logical channel to an ACD group enables your application to receive call status information for all calls associated with the ACD groups, and direct incoming calls to the appropriate destination.

- Monitor channels

A monitor channel is a single channel that you can use to monitor multiple devices. By assigning to this logical entity, your application can receive call data, status, and party information for multiple devices on one channel. Your application can use this information, for example, to record statistics for a number of agents.

- CDN route points

A Control Directory Number (CDN) is a special DN to which you can assign call-handling parameters rather than agents. You route incoming calls to a CDN where the host application determines the routing of the call to a specific destination and how the call is treated whilst waiting in an ACD queue.

1.2 Introduction to CTC/CMP Routines

There are two groups of CTC/CMP routines:

- Routines that control the logical channel
- Routines that perform telephony functions

1.2.1 Routines That Control the Logical Channel

This group of routines gives you control of the logical channel between the user's process and a specific telephony device. You can:

- Assign and deassign logical channels to and from a device (a telephone set, agent position, ACD group, or monitor channel)
- Set and query certain characteristics for a device
- Monitor calls associated with a device
- Monitor telephony on a channel

Table 1–1 shows the functions available for creating and controlling a logical channel, and the routines that provide those functions.

Table 1–1 Controlling the Communications Channel

Communication Function	Routine
Assign a communications channel to a telephone set, agent position, route point, or ACD group and identify the channel uniquely to the application, or assign a monitor channel so that the application monitors a number of devices on a single channel.	ctcCmpAssign
Deassign a channel from its associated device, and release resources associated with the channel.	ctcCmpDeassign
Set monitoring on for a device so that information about that device is returned on a monitor channel.	ctcCmpAddMonitor
Stop monitoring a device on a monitor channel.	ctcCmpRemoveMonitor
Receive call routing requests from the switch so that you can decide where to route calls.	ctcCmpGetRouteQuery
Return information about an ACD group to which a channel is assigned.	ctcCmpGetACDGroupStatus

Table 1–1 Controlling the Communications Channel (Continued)

Communication Function	Routine
Return information about the communications channel and the device to which the channel is assigned.	ctcCmpGetChannelInformation
Return the monitoring state of the assigned device.	ctcCmpGetMonitor
Return information on telephone calls associated with the assigned device. Depending on the type of call, you can receive information on: <ul style="list-style-type: none">• Call events• Other parties involved in the telephone call	ctcCmpGetEvent
Return information about condition values.	ctcCmpErrMsg
Set the monitoring state of the assigned device on or off. Use this routine with the ctcCmpGetEvent routine to receive information on the state of calls associated with a device.	ctcCmpSetMonitor

These routines are described in detail in Chapter 2.

1.2.2 Routines That Perform Telephony Functions

Table 1–2 shows telephony functions available with the CTC/CMP API, 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.	ctcCmpMakeCall
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.	ctcCmpConsultationCall
Complete a transfer call initiated by CtcCmpConsultationCall, and disconnect the assigned device.	ctcCmpTransferCall
Merge two or more calls into a single conference call.	ctcCmpConferenceJoin
Disconnect a consultation call and reconnect a held call.	ctcCmpReconnectHeld

Table 1–2 Telephony Functions (Continued)

Telephony Function	Routine
Respond to a route request from the switch, providing a new destination for the call.	ctcCmpRespondToRouteQuery
Return information about the current setting for the message waiting indicator.	ctcCmpGetMessageWaiting
Set the status for an agent so they can log on or log off as an agent and set the agent mode (for example, “ready to take calls”).	ctcCmpSetAgentStatus
Enable or disable routing by the CTC/CMP server for the assigned route point.	ctcCmpSetRoutingEnable
Apply a treatment to a call which is queued to a route point.	ctcCmpGiveTreatment
Answer an incoming call on the assigned device.	ctcCmpAnswerCall
Clear the active call on the assigned device.	ctcCmpHangupCall
Put the current call on the assigned device on hard hold.	ctcCmpHoldCall
Resume a call that was placed on hold using the ctcCmpHoldCall routine.	ctcCmpResumeCall
Hang up the call for all parties.	ctcCmpClearCall
Return information for the call at the assigned telephony device.	ctcCmpSnapshot

These routines are described in detail in Chapter 2.

1.3 Sequence for Calling CTC/CMP API Routines

To establish and control a logical channel to a device, and to receive data about calls at that device, call CTC/CMP routines in the following sequence:

1. ctcCmpAssign to assign the channel to a device.
2. ctcCmpSetMonitor to set monitoring on for that channel. (Unless you are assigning to a monitor channel. Monitoring is automatically set on when you assign to a monitor channel, so you do not need to use ctcCmpSetMonitor.)
3. ctcCmpGetEvent to monitor the channel and device.

Following this sequence, you can call any of the other CTC/CMP API routines shown in Tables 1–1 and 1–2. For example, if you want to route a call, you call

the `ctcCmpGetRouteQuery` routine. If any of these routines are not successful, you can use `ctcCmpErrMsg` to interpret the returned value.

1.4 Format of Routines

Each routine description in Chapter 2 shows the format of the routine, written in C, and includes a summary of the arguments passed to the routine.

1.5 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.

1.5.1 Data Type

When a calling program passes an argument to a CTC/CMP 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. This can be:

- Byte (unsigned) — 8 bits
- Word (unsigned) — 16 bits
- Integer (unsigned) — 32 bits
- Character string
- Structure
- `ctcChanId`

The structure and `ctcChanId` data types are described in the following subsections.

Data Structures

Some CTC/CMP 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/CMP definitions file installed on your system (see Section 1.6 for more information about definitions files).

There are two types of structure:

- An input structure requires the application to pass information to the CTC/CMP API for one or more of the defined fields. For example, the `ctcCmpAssign` routine requires the application to provide the number of a

telephone device. CTC/CMP 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/CMP the address of a block of memory for the structure. CTC/CMP writes information into the structure for the application to read. CTC/CMP has write-only access to the content of an output structure.

1.5.1.1 CTC/CMP Data Type Definitions

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

Table 1–3 CTC/CMP Data Types

Data Type	Description
ctcChanData	A structure used to pass information about the assigned channel.
ctcChanId	A pointer to a fixed structure used to identify the assigned channel.
ctcCmpApplString	A character string that contains application data, for example, customer reference information. The maximum length of the string is specified by the literal <code>ctcCmpAppDataLen</code> defined in a CTC/CMP definitions file.
ctcCmpAssignData	A structure used to pass information required to create a channel to a device.
ctcCmpCodeString	A null terminated string containing authorization and account code digits. The maximum length for <code>ctcCmpCodeString</code> is defined by the literal <code>ctcCodeLen</code> .
ctcCmpEventData	A structure used to pass information relating to an event on the assigned channel.
ctcCmpGetACDData	A structure containing statistical information on an ACD group.
ctcCmpNetCallID	A structure containing the unique call reference identifier.
ctcCmpRouteData	A structure containing information relating to a call presented to the application for routing.
ctcDeviceString	A character string usually containing the number for a device, for example, a DN. The maximum length for <code>ctcDeviceString</code> is specified by the literal <code>ctcMaxDnLen</code> defined in a CTC/CMP definitions file.

Table 1–3 CTC/CMP Data Types (Continued)

Data Type	Description
ctcLogIdString	A null-terminated character string containing the logical identifier for the link between the CTC/CMP server and the switch.
ctcNameString	A null-terminated character string containing the network name or address for the CTC/CMP server. The maximum length for ctcNameString is specified by the literal ctcCallerNameLen defined in a CTC/CMP definitions file.
ctcNetString	A null-terminated character string that identifies the network protocol used between the CTC/CMP client and CTC/CMP server. The maximum length for ctcNetString is specified by the literal ctcNetLen defined in a CTC/CMP definitions file.
ctcTimeStamp	A structure containing details of the time an event or route request occurred.

1.5.2 Access to Data

The *access* entry indicates whether the CTC/CMP routine:

- 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)

1.5.3 Passing Mechanism

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

- **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 an address of 2000, which currently contains the value 100, the argument entry will contain 2000.

1.5.4 Passing Optional Data

For some arguments, passing data is optional. This means that you 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.

The data that you specify depends on the passing mechanism for the argument:

- 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/CMP, 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/CMP, 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.

1.6 Definitions

During the CTC/CMP installation procedure, CTC/CMP copies a definitions file, `ctcmpdef.h`, to your system for you to include in your C program. This file includes the following definitions files for condition values, constants, and data types:

These definitions...	Are in these files...
Condition Values for Status Returns	<code>ctcmperr.h</code> and <code>ctc_err.h</code>
Constants	<code>ctcmpcod.h</code> and <code>ctc_code.h</code>
Data Types (described in Table 1–3)	<code>ctcmprpc.h</code>

The file `ctcmpdef.h` and the definitions files that it includes are copied to the directory specified during installation. For more information, refer to the *CT-Connect for CompuCALL Installation and Administration Guide*.

1.7 Constants

CTC/CMP constants have one of the following prefixes:

This prefix...	Is used for...
ctcK_	Literals. For example, the value <code>ctcK_AgentReady</code> can be specified with the <code>agentMode</code> argument for the <code>ctcCmpSetAgentStatus</code> routine to indicate that the agent is ready to take calls.
ctcM_	Function-supported masks. These values are used to indicate whether a function is supported. For example, the value <code>ctcM_Assign</code> can be returned in the <code>procedureSupport</code> field for the <code>ctcCmpChanData</code> structure. (See the description of the <code>ctcCmpGetChannelInformation</code> routine for more information.)

1.8 Errors and Condition Values for Status Returns

Each 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 in your application you always check the returned status to determine the success or failure of calls to CTC/CMP routines, and choose a suitable recovery path if there is an error.

The descriptions of routines in Chapter 2 show the possible return values for each routine, and you can see a description of all CTC/CMP return values in Chapter 3.

1.8.1 Link Problems

If the link between the DMS-100 and CTC/CMP server fails, the CTC/CMP server:

- Returns a `ctcLinkDown` condition value
- Clears all monitors and cancels any outstanding `ctcCmpGetEvent` and `ctcCmpGetRouteQuery` requests
- Deassigns all channels

1.9 Exception Handling

A severe network problem that affects communication between the CTC/CMP client and CTC/CMP server may result in a software exception. If you want to handle this type of exception, refer to the Remote Procedure Call (RPC) programming documentation for your operating system.

1.10 Calling CTC/CMP Routines

All CTC/CMP 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 `ctcCmpGetEvent` 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 the 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.

1.11 CTC/CMP and Multithreaded Programming

Sections 1.11.1 through 1.11.5 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.

1.11.1 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.

1.11.2 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.

1.11.3 Thread Execution

A processor executes a thread until the thread has to wait for a resource to become available, for example, 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.

1.11.4 Using Multithreaded Programming with CTC/CMP

Using multithreaded programming, a CTC/CMP 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 `ctcCmpMakeCall` routine.
- It can create another thread for monitoring the device. For example, for calling the `ctcCmpGetEvent` routine which returns only when there is call activity. Dialogic recommends that you create a separate thread for this routine.

An online programming example is provided as part of the CTC/CMP API kit. This example shows how multithreaded programming is used. For details of its location, refer to the *CT-Connect for CompuCALL Installation and Administration Guide*.

1.11.5 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 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 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 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.

1.12 Compiling and Linking Your Program

Sections 1.12.1 to 1.12.7 contain platform-specific information about compiling and linking your program.

1.12.1 Digital UNIX Client

On a CTC/CMP client running Digital UNIX, the CTC/CMP API is provided as the shareable object, `libctcmpapi.sl`. 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 ctcmpprog.c
```

where `ctcmpprog.c` is source code written in C.

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

```
# cc -o ctcmpprog ctcmpprog.o -lctcmpapi -ldce -lpthreads -lc_r \  
-lmach
```

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

1.12.2 HP-UX Client

On a CTC/CMP client running HP-UX, the CTC/CMP API is provided as the shareable library, `libctcmpapi.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 ctcmpprog.o -Ae +04 -I/usr/include/reentrant \  
-D_REENTRANT ctcmpprog.c
```

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

To link a program written in C, you use:

```
# cc -s -o ctcmpapp ctcmpprog.o -Wl,-Bimmediate,-Bnonfatal \  
-lctcmpapi -ldce
```

where *ctcmpprog.o* is the compiled program and *ctcmpapp* is the executable image.

1.12.3 OpenVMS Client

On a CTC/CMP client running OpenVMS, the CTC/CMP API is provided as the shareable image, `SYS$SHARE:CTCMPAPI.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:CTCMPAPI.EXE/SHAREABLE
```

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

2. Use the `LINK` command to link your image:

```
$ LINK ctcmp_program, filename.OPT/OPTION, DCE:DCE.OPT/OPTION
```

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

1.12.4 OS/2 Client

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

```
icc ctcmpprog.c /Gm+ /Su4 /Ms /C+ /Sem -D_CMS_PROTO_  
-D_CMA_NOWRAPPERS_ -DCMA_UNIPROCESSOR -DINTEL80x86 -IDBMOS2
```

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

The following example shows how to link the CTC/CMP program:

```
ilink ctcmpprog.obj /E /NOI /NOE /ST:100000 /O:ctcmpprog  
ctcmpapi.lib dceos2.lib os2386.lib
```

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

1.12.5 SCO OpenServer Client

To compile your program on a CTC/CMP client running SCO OpenServer, you use:

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

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

To link your program, you use:

```
# cc -ctcmpprog.o /lib/crt0.o -o ctcmpapp -s -lctcmpapi -ldce -lcma  
\ -lm -lsocket -lc
```

where *ctcmpprog.o* is the compiled program and *ctcmpapp* is the executable image.

1.12.6 Solaris Client

On a CTC/CMP client running Solaris software, the CTC/CMP API is provided as the shareable object, *libctcmpapi.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 ctcmpprog.c
```

where *ctcmpprog.c* is CTC/CMP source code written in C.

The following example shows how to link the program:

```
# cc -o ctcmpprog -s ctcmpprog.o -lctcmpapi -ldce
```

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

1.12.7 Windows NT and Windows 95 Clients

Before you compile and link your program on a Windows NT or Windows 95 system, note the information in the following subsections.

Multithreaded Programs and Thread Stack Size

If you create threads for the `ctcCmpGetEvent` routine in your program, note the following:

- 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.
- 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.

Paths

During the CTC/CMP API installation, the file `CTCVARS.BAT` is copied to the *drive:\directory\bin* directory. This file contains the following paths:

```
drive:\directory\lib  
drive:\directory\include
```

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

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

1.13 Changes to CTC/CMP for Version 3.0

Table 1–4 provides a summary of new features and routines provided with this version of CTC/CMP.

Table 1–4 Summary of Changes to CTC/CMP for V3.0

This version of CTC/CMP adds support for...	For more information, refer to...
CTC/CMP client on the Sun Solaris platform.	The following: <ul style="list-style-type: none"> For how to install the Solaris client, the <i>CT-Connect for CompuCALL Installation and Administration Guide</i>. For how to compile and link a program on the Solaris platform, Section 1.12.6.
TCP/IP links between the CTC/CMP server and the DMS-100.	For details of how to configure TCP/IP links, the <i>CT-Connect for CompuCALL Installation and Administration Guide</i> .
Additional API routines: ctcCmpAnswerCall ctcCmpClearCall ctcCmpGetMessageWaiting ctcCmpGiveTreatment ctcCmpHangupCall ctcCmpHoldCall ctcCmpResumeCall ctcCmpSetRoutingEnable ctcCmpSnapshot	For details of how to use the routines, Chapter 2.
Additional events: ctcK_CmpAgentInformation ctcK_AgentMode ctcK_CmpDeviceStatus ctcK_CmpMessageWaiting ctcK_CmpTreatmentComplete ctcK_OffHook ctcK_OpConferenced ctcK_TpConferenced ctcK_TpRetrieved ctcK_TpTransferred	For details of how events are returned, Chapter 2.
The CDN Route Point device type.	For details of how the device types are used in routines, Chapter 2.

Table 1–4 Summary of Changes to CTC/CMP for V3.0 (Continued)

This version of CTC/CMP adds support for...	For more information, refer to...
Improved link failure detection. The CTC/CMP server can periodically poll the link to the switch, checking whether it is still enabled. If the switch does not acknowledge the poll requests from the CTC/CMP server, the CTC/CMP server assumes that the link has failed and restarts it.	The following: <ul style="list-style-type: none">• For details of how the CTC server reports link problems to the CTC API, see Section 1.8.1.• For details of how to use the Configuration Program to enable link state checking, see the <i>CT-Connect for CompuCALL Installation and Administration Guide</i>.
Altering the number of buffers you allocate to hold recent device and monitor channel events.	For details of how to alter the number of buffers allocated, see the <i>CT-Connect for CompuCALL Installation and Administration Guide</i> .

1.14 Compatibility With Previous Versions of CTC/CMP

If you are upgrading from CTC/CMP Version 1.1, note the following:

- CTC/CMP clients running Version 1.1 of the CTC/CMP API are compatible with a CTC/CMP server running Version V3.0 of the CTC/CMP server software.
- You can continue to use CTC/CMP Version 1.1 applications on a client system running CTC/CMP API Version V3.0 software.

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

The only exception to this backwards compatibility is with the `ctcCmpRespondToRouteQuery` routine. The Version 3.0 implementation of this routine contains an additional parameter (`numberType`). If your existing application uses `ctcCmpRespondToRouteQuery`, you will have to:

- Modify your code so that it matches the description in Chapter 2.
- Recompile your application before it can be used with CTC/CMP Version 3.0.

Note that when you use `ctcCmpAssign`, the `APIversion` value `ctcK_CurrentVersion` is the equivalent of `ctcK_CmpCTCV30` for this version of CTC/CMP. Programs compiled with `ctcK_CurrentVersion` or `ctcK_CmpCTCV30`

can use the CTC/CMP API routines and events new to CTC/CMP V3.0 (as listed in Section 1.13). For more information about `ctcCmpAssign`, refer to Chapter 2.

Routine Specifications

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

ctcCmpAddMonitor

ctcCmpAddMonitor

Adds a Device to a Monitor Channel

Format in C

```
unsigned int ctcCmpAddMonitor  
            (ctcChanId          channel,  
             ctcCmpAssignData *assignData)
```

Description

The `ctcCmpAddMonitor` routine sets monitoring on for a device and associates it with a monitor channel. A monitor channel is a single channel used to monitor multiple devices, such as, telephones or agent positions. Event information for the device is returned on the monitor channel.

Monitoring on a Monitor Channel

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

1. `ctcCmpAssign` to create a monitor channel
2. `ctcCmpAddMonitor` for each device you want to monitor on the monitor channel
3. `ctcCmpGetEvent` 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.

Recommendation

If you are monitoring a number of devices on the monitor channel and activity on each device is high, a number of events may occur at the same time. In this situation, it is possible for the CTC/CMP server to lose an event message and return a `ctcEventDataLost` error.

To prevent this happening, Dialogic recommends that you spread the monitoring of these high activity channels over several monitor channels.

Monitoring Another Monitor Channel

To monitor another monitor channel, use the `ctcCmpGetChannelInformation` routine 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.

ctcCmpAddMonitor

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 `ctcCmpRemoveMonitor`. For more information, see the description of the `ctcCmpRemoveMonitor` routine.

Restrictions

The following restrictions apply:

- You cannot monitor a primary or supplementary DN for an ACD group on a monitor channel.
- This routine is supported for monitor channels only.

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 `ctcCmpAssign` 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/CMP* definitions file (see Section 1.6).

ctcCmpAddMonitor

assignData

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpAssignData`. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

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

The following sections describe the `ctcCmpAssignData` fields:

- **deviceType**

This 16-bit field identifies the type of device you are assigning to the monitor channel.

Specify one of the following values in the `deviceType` field:

Specify this value...	To monitor this type of device...
<code>ctcK_Dn</code>	Telephony device
<code>ctcK_RoutePoint</code>	CDN route point
<code>ctcK_AgentPosition</code>	Agent position
<code>ctcK_MonitorChannel</code>	Monitor channel

- **APIversion**

This 8-bit field identifies the version of CTC/CMP API used. Specify one of the following values:

Value	Description
<code>ctcK_CmpCTCV30</code>	Specify this value if you are writing a CTC/CMP application for use only with Version 3.0 of the CTC/CMP API software.
<code>ctcK_CurrentVersion</code>	Specify this value and your application will be compatible with the current version of the CTC/CMP API installed on your CTC/CMP client system. When you recompile and upgrade to a future version of the CTC/CMP API, your application will automatically gain access to any new events provided as part of that CTC/CMP API.

ctcCmpAddMonitor

To ensure compatibility with future versions of the CTC/CMP API, Dialogic recommends you use `ctcK_CurrentVersion`. Values that identify specific versions of the CTC/CMP API may not be available with future releases of CTC/CMP.

- **APIextensions**

CTC/CMP does not use the data in this field. Specify the value zero.

- **deviceDN**

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

- For a telephony device, primary DN for an ACD group, or supplementary DN for an ACD group, use this field to specify its directory number (telephone number). This ASCII string can contain any combination of numbers 0 through 9 and the characters * and #. For a telephony device the string is 10 digits. For example, 9123876543.
- For a CDN route point, specify the ten digit CDN number. For example, 5127790401.
- For an agent position, use this field to specify the agent position number. This ASCII string can contain up to 4 digits and any combination of numbers 0 through 9. For example, 6573.
- For a monitor channel, specify the device number returned in the `setDN` field of the `ctcChanData` structure. This is returned when you call `ctcCmpGetChannelInformation` for the monitor channel. See the description of `ctcCmpGetChannelInformation` for more information.

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 #. Note that you must specify the device number exactly as it is returned in the `setDN` field, using uppercase for letters.

The maximum length for `deviceDN` is specified by the literal `ctcMaxDnLen` in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

ctcCmpAddMonitor

The DMS-100 can return the following condition values for this routine:

- ctcNotAllowed
- ctcInvDN
- ctcAssociated
- ctcOverallMonLimEx
- ctcResourceBusy

For details of the condition values that can be returned by the CTC/CMP API and CTC/CMP server, refer to Chapter 3.

ctcCmpAnswerCall

Answer a Call

Format in C

```

unsigned int ctcCmpAnswerCall
            (ctcChanId          *channel,
             ctcCmpNetCallId    *callRefId)

```

Description

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

This routine is useful for operation on a phone with a hands-free capability because the user can answer the telephone without lifting the handset. However, it cannot be used with standard telephones (2500 sets).

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

1. You have set monitoring on, using `ctcCmpSetMonitor`.
2. You are using the `ctcCmpGetEvent` routine, which indicates a change in state to receive (ringing).

The call reference identifier information for the answered call is returned by the DMS-100.

Restrictions

This routine is not supported for channels assigned to ACD groups, CDNs, or monitor channels. In addition, it is not available for 500/2500 sets.

Arguments

channel
 type: **ctcChanId**
 access: **write only**
 mechanism: **by value**

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

The `ctcChanId` datatype is defined in a CTC/CMP definitions file (see Section 1.6).

ctcCmpAnswerCall

callRefId

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpNetCallId`. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
ctcCmpNetCallId {  
    unsigned int    netNodeId;  
    unsigned int    localCallId;  
};
```

Information in the `ctcCmpNetCallId` structure provides a unique call reference identifier for the conference call. The DMS-100 returns the following information in the `ctcCmpNetCallId` structure:

- **netNodeId**

This 32-bit field contains a network node identifier. This is used to identify which DMS-100 first receives a call when the call is handled by more than one DMS-100.

- **localCallId**

This field returns a local call identifier. This is used to uniquely identify the call on the DMS-100.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following condition values for this routine:

```
ctcBadObjState  
ctcInvParam  
ctcMissingParam  
ctcNotAllowed  
ctcUnavailable
```

For details of the condition values that can be returned by the CTC/CMP API and CTC/CMP server, refer to Chapter 3.

ctcCmpAssign

Assign a Channel

Format in C

```

unsigned int ctcCmpAssign
            (ctcChanId          *channel,
             ctcCmpAssignData *assignData,
             ctcNameString    serverName,
             ctcLogIdString   logId,
             ctcNetString     network)

```

Description

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

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

You can use `ctcCmpAssign` to assign a channel to the following:

- Individual telephone (Centrex, residential, or secondary DN)
- Agent position
- Primary DN for an ACD group
- CDN route point
- Supplementary DN for an ACD group
- Monitor channel

When to Use `ctcCmpAssign`

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

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.

ctcCmpAssign

Monitor Channels

If you need to monitor a number of devices, you can use `ctcCmpAssign` 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. `ctcCmpAssign` to assign a monitor channel
2. `ctcCmpAddMonitor` for each device you want to monitor on the monitor channel
3. `ctcCmpGetEvent` to return information for all devices associated with the monitor channel

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

If you need to monitor a number of devices on a monitor channel and activity on each device is high, a number of events may occur at the same time. In this situation, it is possible for the CTC/CMP server to lose an event message and return a `ctcEventDataLost` error.

To prevent this happening, Dialogic recommends that you spread the monitoring of these high activity devices over a number of monitor channels.

Supported Devices and Routines

Table 2–1 shows which routines are supported by each type of device.

Table 2–1 Routines Supported by Device Type

Routine	Telephone	Agent Position	ACD Group	Monitor Channel	CDN Route Point
<code>ctcCmpAddMonitor</code>				X	
<code>ctcCmpAnswerCall</code>	X†	X†			
<code>ctcCmpAssign</code>	X	X	X	X	X‡
<code>ctcCmpClearCall</code>	X†	X†			
<code>ctcCmpConferenceJoin</code>	X†	X			
<code>ctcCmpConsultationCall</code>	X†	X			

† NA006 or equivalent

‡ NA008 or equivalent

ctcCmpAssign

Table 2–1 Routines Supported by Device Type (Continued)

Routine	Telephone	Agent Position	ACD Group	Monitor Channel	CDN Route Point
ctcCmpDeassign	X	X	X	X	X‡
ctcCmpErrMsg	X	X	X	X	X‡
ctcCmpGetACDGroupStatus			X		
ctcCmpGetAgentStatus		X#			
ctcCmpGetChannelInformation	X	X	X	X	X‡
ctcCmpGetEvent	X	X	X	X	X‡
ctcCmpGetMessageWaiting	X†				
ctcCmpGetMonitor	X	X	X		
ctcCmpGetRouteQuery			X*		X‡
ctcCmpGetRoutingEnable					X#
ctcCmpGiveTreatment					X‡
ctcCmpHangupCall	X†	X†			
ctcCmpHoldCall	X†	X†			
ctcCmpMakeCall	X†	X			
ctcCmpReconnectHeld	X†	X			
ctcCmpRemoveMonitor				X	
ctcCmpRespondToRouteQuery			X*		X‡
ctcCmpResumeCall	X†	X†			
ctcCmpSetAgentStatus		X			
ctcCmpSetMonitor	X	X	X		X‡
ctcCmpSetRoutingEnable					X‡
ctcCmpSnapshot	X†				
ctcCmpTransferCall	X†	X			

† NA006 or equivalent

‡ NA008 or equivalent

NA009 or equivalent

* Only if assigned to a primary DN for an ACD group

ctcCmpAssign

Arguments

channel

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

This argument is a pointer that receives the address of a `ctcChanId` datatype. The datatype is defined in a CTC/CMP definitions file (see Section 1.6).

You use the channel ID with all other CTC/CMP routines to identify the device.

assignData

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpAssignData`.

You use the `assignData` argument to identify:

- The type of device to which you want to assign a channel.
- The version of CTC/CMP API used.
- The number for the device.

The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
ctcCmpAssignData {
    unsigned short    deviceType;
    unsigned char     APIversion;
    unsigned char     APIextensions;
    unsigned char     deviceDN [ctcMaxDnLen];
};
```

The following subsections describe the contents of `ctcCmpAssignData` fields:

- **deviceType**

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

- Telephones associated with 10-digit telephone numbers, such as, a Centrex DN or a secondary DN.
- An agent position.
- A primary DN for an ACD group.

- A CDN route point
- A supplementary DN for an ACD group.
- A monitor channel. This is a logical entity that enables you to create a single channel to monitor multiple telephony devices, agent positions, groups, and other monitor channels.

Specify one of the following values in the deviceType field:

Specify this value...	To assign to...
ctcK_Dn	A telephony device
ctcK_RoutePoint	A CDN route point
ctcK_AgentPosition	An agent position
ctcK_PrimaryAcid	A primary DN for an ACD group
ctcK_SupplementaryAcid	A supplementary DN for an ACD group
ctcK_MonitorChannel	A monitor channel

- **APIversion**

This 8-bit field identifies the version of CTC/CMP API used. Specify one of the following values:

Value	Description
ctcK_CmpCTCV30	Specify this value if you are writing a CTC/CMP application for use only with Version 3.0 of the CTC/CMP API software.
ctcK_CurrentVersion	Specify this value and your application will be compatible with the current version of the CTC/CMP API installed on your CTC/CMP client system. When you recompile and upgrade to a future version of the CTC/CMP API, your application will automatically gain access to any new events provided as part of that CTC/CMP API.

To ensure compatibility with future versions of the CTC/CMP API, Dialogic recommends you use ctcK_CurrentVersion. Values that identify specific versions of the CTC/CMP API may not be available with future releases of CTC/CMP.

- **APIextensions**

CTC/CMP does not use the data in this field. Specify the value zero.

ctcCmpAssign

- **deviceDN**

This field identifies the device to which a channel is assigned:

- For a telephony device, primary DN for an ACD group, or supplementary DN for an ACD group, use this field to specify its directory number (telephone number). This ASCII string can contain any combination of numbers 0 through 9 and the characters * and #. For a telephony device, the string is 10 digits. For example, 9123876543.
- For a CDN route point, specify the ten digit CDN number. For example, 5127790401.
- For an agent position, use this field to specify the ACD position identifier. This ASCII string can contain up to 4 digits and any combination of numbers 0 through 9 and the characters * and #. For example, 6573.
- For a monitor channel, specify a zero-length character string.

The maximum length for deviceDN is specified by the literal ctcMaxDnLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

serverName

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

This string can contain the name or address of the CTC/CMP server.

The maximum length for serverName is specified by the literal ctcNodeNameLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

logId

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/CMP server and is defined either during the installation of the CTC/CMP server software, or after the installation with the CTC/CMP Configuration Program (see the *CT-Connect for CompuCALL Installation and Administration Guide*).

The maximum length for logicalIdentifier is specified by the literal ctcLogIdLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

network

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/CMP client and the CTC/CMP server.

Check with the system manager of your CTC/CMP network for details of the network protocol, and 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
Local RPC	ncalrpc

Local RPC can be used if the CTC/CMP client and the CTC/CMP server are the same system.

The maximum length for networkType is specified by the literal ctcNetLen in the CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value ctcSuccess. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following condition values for this routine:

- ctcNotAllowed
- ctcInvDN
- ctcAssociated
- ctcOverallMonLimEx
- ctcResourceBusy

ctcCmpAssign

For details of the condition values that can be returned by the CTC/CMP API and CTC/CMP server, refer to Chapter 3.

ctcCmpClearCall

Clear Call on All Parties

Format in C

```
unsigned int ctcCmpClearCall
            (ctcChanId    channel,
             ctcCmpNetCallId *callRefId)
```

Description

The ctcCmpClearCall routine hangs up the call for all parties, for example in a conference call.

The call reference information is returned by the DMS-100.

Restrictions

This routine is not supported for channels assigned to ACD groups, CDNs, or monitor channels.

After a ctcCmpClearCall, all users at a 500/2500 set in the conference call receive the dial tone. Further calls can then be made. However, if a user stays off-hook for too long, the set will receive a lockout treatment.

Arguments

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

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

ctcCmpClearCall

callRefId

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpNetCallId`. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
ctcCmpNetCallId {  
    unsigned int    netNodeId;  
    unsigned int    localCallId;  
};
```

Information in the `ctcCmpNetCallId` structure provides a unique call reference identifier for the conference call. The DMS-100 returns the following information in the `ctcCmpNetCallId` structure:

- **netNodeId**

This 32-bit field contains a network node identifier. This is used to identify which DMS-100 first receives a call when the call is handled by more than one DMS-100.

- **localCallId**

This field returns a local call identifier. This is used to uniquely identify the call on the DMS-100.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following condition values for this routine:

```
ctcBadObjState  
ctcInvParam  
ctcMissingParam  
ctcNotAllowed  
ctcUnavailable
```

For details of the condition values that can be returned by the CTC/CMP API and CTC/CMP server, refer to Chapter 3.

ctcCmpConferenceJoin

Merge Calls into a Conference

Format in C

```

unsigned int ctcCmpConferenceJoin
            (ctcChanId          channel,
             ctcCmpNetCallId   *callRefId)

```

Description

The `ctcCmpConferenceJoin` 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 can:

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

For more information about making a conference call, refer to the description of `ctcCmpConsultationCall`.

The call reference information for the conference call is returned by the DMS-100.

Restriction

This routine is not supported for channels assigned to ACD groups, CDNs, or monitor channels.

Arguments

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

This argument is a `ctcChanId` datatype that contains the channel ID value returned by `ctcCmpAssign` for the device in use.

ctcCmpConferenceJoin

callRefId

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpNetCallId`. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
ctcCmpNetCallId {  
    unsigned int    netNodeId;  
    unsigned int    localCallId;  
};
```

Information in the `ctcCmpNetCallId` structure provides a unique call reference identifier for the conference call. The DMS-100 returns the following information in the `ctcCmpNetCallId` structure:

- **netNodeId**

This 32-bit field contains a network node identifier. This is used to identify which DMS-100 first receives a call when the call is handled by more than one DMS-100.

- **localCallId**

This field returns a local call identifier. This is used to uniquely identify the call on the DMS-100.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following condition values for this routine:

```
ctcBadObjState  
ctcInvParam  
ctcMissingParam  
ctcNotAllowed  
ctcNotLoggedIn  
ctcUnavailable
```

For details of the condition values that can be returned by the CTC/CMP API and CTC/CMP server, refer to Chapter 3.

ctcCmpConsultationCall

Make a Consultation Call

Format in C

```

unsigned int ctcCmpConsultationCall
            (ctcChanId          channel,
             ctcDeviceString   calledNumber,
             unsigned int     consultType,
             ctcCmpNetCallId  *callRefId)

```

Description

The `ctcCmpConsultationCall` 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:

- `ctcCmpTransferCall` to transfer the call and disconnect the assigned device
- `ctcCmpConferenceJoin` 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 `ctcCmpConsultationCall`. This routine makes the telephone call and allows the switch to allocate any required resources.

The call reference information for the consultation call is returned by the DMS-100.

Making a Conference Call

To create a conference call, you make the initial call using `ctcCmpMakeCall` and `ctcCmpConsultationCall` for the second call.

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

1. A calls B, using `ctcCmpMakeCall`, and B answers.
2. A calls C, using `ctcCmpConsultationCall`, which automatically puts the call to B on consultation hold.
3. A invokes `ctcCmpConferenceJoin` when connected and talking to C. A, B, and C are then in a conference call.

Note that the DMS-100 supports 3-way conference calls only.

ctcCmpConsultationCall

Transferring a Call

To transfer a call, use `ctcCmpConsultationCall` followed by `ctcCmpTransferCall`.

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

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

Restriction

This routine is not supported for channels assigned to ACD groups, CDNs, or monitor channels.

Arguments

channel

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

This argument is a `ctcChanId` datatype that contains the channel ID value returned by `ctcCmpAssign` 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/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

ctcCmpConsultationCall

consultType

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

This argument specifies the type of consultation call made to the third party. Use one of the following values:

Specify this value...	To initiate a...
ctcK_ConsultTransfer	Call transfer.
ctcK_ConsultConference	Conference call.

callRefId

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpNetCallId`. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
struct ctcCmpNetCallId {
    unsigned int    netNodeId;
    unsigned int    localCallId;
}
```

Information in the `ctcCmpNetCallId` structure provides a unique call reference identifier for the call. The DMS-100 returns the following information in the fields of the `ctcCmpNetCallId` structure:

- **netNodeId**

This 32-bit field contains a network node identifier. This is used to identify which DMS-100 first receives a call when the call is handled by more than one DMS-100.

- **localCallId**

This field returns a local call identifier. This is used to uniquely identify the call on the DMS-100.

ctcCmpConsultationCall

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following condition values for this routine:

- `ctcBadObjState`
- `ctcInvParam`
- `ctcMissingParam`
- `ctcNotAllowed`
- `ctcNotLoggedIn`
- `ctcUnavailable`

For details of the condition values that can be returned by the CTC/CMP API and CTC/CMP server, refer to Chapter 3.

ctcCmpDeassign Deassign a Channel

Format in C

unsigned int **ctcCmpDeassign** (*ctcChanId* channel)

Description

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

Use this routine at the end of a user session; that is, when the user has finished using a CTC/CMP application and the application no longer needs to make use of the device to which the channel was assigned.

Monitoring and routing is switched off before `ctcCmpDeassign` completes. If you call `ctcCmpDeassign` and there are outstanding monitoring or routing requests, a condition value is returned:

- For `ctcCmpGetEvent`, the `ctcMonitorOff` condition value is returned.
- For `ctcCmpGetRouteQuery`, 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 ID value returned by `ctcCmpAssign` for the device in use.

ctcCmpErrMsg

ctcCmpErrMsg

Get the Defined Name for a Condition Value

Format in C

```
char *ctcCmpErrMsg (unsigned int    errorCode)
```

Description

The `ctcCmpErrMsg` routine returns the defined name associated with a CTC/CMP 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/CMP conditions.

Each condition value is associated with a name in a CTC/CMP error definitions file (see Section 1.6 for details of the `ctcmperr.h` and `ctc_err.h` definitions files). When you use `ctcErrMsg`, CTC/CMP 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/CMP 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`.

Note, there are no condition values returned for the `ctcErrMsg` routine.

Arguments

errorCode

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

This 32-bit integer contains the condition value returned by a CTC/CMP routine.

CTC/CMP returns the address of a null-terminated character string that contains the name associated with the condition value that you specify.

Note that:

- 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.

ctcCmpErrMsg

- If you specify a condition value for an RPC error, the character string contains:
 - The name defined by CTC/CMP associated with the condition value
 - The RPC name (with the prefix `rpc_`) associated with the condition value

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

ctcCmpGetACDGroupStatus

ctcCmpGetACDGroupStatus

Get Information About an ACD Group

Format in C

```
unsigned int ctcCmpGetACDGroupStatus
            (ctcChanId          channel,
             ctcCmpGetACDData  *queueData)
```

Description

This routine returns information about an ACD group to which a channel is assigned. The information includes:

- Time set on the switch when the information was collected
- State of the ACD group (for example, accepting calls)
- Static settings for the ACD group (for example, the maximum number of incoming calls that can be queued)
- Current status information (for example, the actual number of incoming calls queued or calls waiting to be transferred)
- Agent status information
- ACD DN associated with the ACD group

Restrictions

This routine is not supported for channels assigned to telephones, CDNs, agent positions, or monitor channels.

Arguments

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

This argument is a `ctcChanId` datatype that contains the channel ID value returned by `ctcCmpAssign` for the device in use.

ctcCmpGetACDGroupStatus

queueData

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

This argument is the address of a fixed-format structure, for which you allocate memory of type `ctcCmpGetACDDate`. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
struct ctcCmpGetACDData {
    unsigned short    hours;
    unsigned short    minutes;
    unsigned short    seconds;
    unsigned short    acdState;
    unsigned short    maxPqSize;
    unsigned short    maxCtqSize;
    unsigned short    acdDnPrio;
    unsigned short    pqSize;
    unsigned short    ctqSize;
    unsigned short    outqSize;
    unsigned short    pq0;
    unsigned short    pq1;
    unsigned short    pq2;
    unsigned short    pq3;
    unsigned short    agentsLgd;
    unsigned short    agentsBsy;
    unsigned short    agentsIdle;
    unsigned short    agentsNr;
    unsigned short    agentsCtq;
    ctcDeviceString   agentsAcDn;
}
```

The following subsections describe these fields.

- **hours, minutes, seconds**

These 16-bit fields contain the exact time that the information was collected on the switch. Information in the hour field is based on the twenty-four hour clock. The minutes and seconds fields can be used to establish the exact time that the information was gathered (as defined on the switch).

ctcCmpGetACDGroupStatus

- **acdState**

This 16-bit field specifies the current operational state of the ACD group. It contains one of the following values:

Value	Description
ctcK_AcceptingCalls	The ACD group is in service and incoming calls are being serviced or queued.
ctcK_Overflow	The ACD group is in overflow state. New calls may be rerouted to another destination, for example, to another ACD group or an announcement (as defined on the DMS-100).
ctcK_NightService	The ACD group is in night service state either because all ACD agents have logged off or because the ACD supervisor has manually set this state. Incoming calls may be rerouted to another destination, for example, to a night service route or an announcement (as defined on the DMS-100).
ctcK_ControlledInterflow	The ACD supervisor has placed the ACD Group in controlled interflow state so that it only services calls that have already been queued. New calls for this ACD group are routed to a Controlled Interflow Route (as defined on the DMS-100).
ctcK_StatusReroute	The ACD group is not servicing or queuing calls because of an equipment fault that affects the telephone lines for agents in the group. New calls are routed to another destination (as defined on the DMS-100). Calls that have already been queued for the group, remain in the queue until they can be answered or treated for exceeding threshold conditions defined on the DMS-100.

- **maxPqSize**

This 16-bit field specifies the maximum number of incoming calls that can be queued for the group. This is defined on the DMS-100 as the maximum number of calls in the Incoming Call Queue.

ctcCmpGetACDGroupStatus

- **maxCtqSize**

This 16-bit field specifies the maximum number of calls that can be queued for transfer. This is defined on the DMS-100 as the maximum number of calls in the Call Transfer Queue.
- **acdDnPrio**

This 16-bit field specifies the priority associated with the assigned device, as defined on the DMS-100.
- **pqSize**

This 16-bit field specifies the total number of calls queued. This includes calls in the:

 - Incoming Calls Queue
 - Call Transfer Queue
 - Overflow Out Queue
- **ctqSize**

This 16-bit field specifies the number of calls queued in the ACD group's Call Transfer Queue.
- **outqSize**

This 16-bit field specifies the number of calls queued in the ACD group's Overflow Out Queue.
- **pq0**

This 16-bit field specifies the number of calls queued at priority 0.
- **pq1**

This 16-bit field specifies the number of calls queued at priority 1.
- **pq2**

This 16-bit field specifies the number of calls queued at priority 2.
- **pq3**

This 16-bit field specifies the number of calls queued at priority 3.
- **agentsLgd**

This 16-bit field specifies the number of agents currently logged in for the assigned agent group.

ctcCmpGetACDGroupStatus

- **agentsBsy**

This 16-bit field specifies the number of agents currently busy (active on a call).

- **agentsIdle**

This 16-bit field specifies the number of agents currently idle (waiting for calls). This should be zero if incoming calls are queued (see `pqSize` field).

- **agentsNr**

This 16-bit field specifies the number of agents who are not ready to take calls. These agents may be wrapping up a call or may be temporarily unavailable.

- **agentsCtq**

This 16-bit field specifies the number of calls waiting in the Call Transfer Queue for transfer to agents. These can be calls that are transferred from one agent to another, for example, or from one ACD group to another.

A call in the Call Transfer Queue is directed to the destination agent as soon as they become available and before they receive any new, incoming calls (queued in the Incoming Calls Queue).

- **agentsAcdDn**

This character string contains the primary DN for the ACD group. The maximum length for `agentsAcdDn` is specified by the literal `ctcMaxDnLen` in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following condition values for this routine:

- `ctcInvParam`
- `ctcMissingParam`
- `ctcNotAllowed`
- `ctcUnavailable`

For details of the condition values that can be returned by the CTC/CMP API or CTC/CMP server, refer to Chapter 3.

ctcCmpGetAgentStatus

Get Information About an Agent

Format in C

```
unsigned int ctcCmpGetAgentStatus (ctcChanId channel)
```

Description

The `ctcCmpGetAgentStatus` routine gets the login or readiness status of an ACD agent. The status is returned by the DMS-100 through a `ctcK_AgentMode` event.

Restrictions

The following restrictions apply:

- This routine requires NA009 (or equivalent) or later.
- This routine is only supported for agent positions.

Arguments

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

This argument is a `ctcChanId` datatype that contains the channel ID value returned by `ctcCmpAssign` for the ACD agent.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the CTC/CMP API or the CTC/CMP server.

The CTC/CMP server can return the following condition values for this routine:

`ctcInvDN`
`ctcInvParam`
`ctcNotAllowed`
`ctcUnavailable`

For details, refer to Chapter 3.

ctcCmpGetChannelInformation

ctcCmpGetChannelInformation Get Information About a Channel

Format in C

```
unsigned int ctcCmpGetChannelInformation  
            (ctcChanId      channel,  
             ctcChanData   *channelData)
```

Description

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

- Line type (telephone, agent position, primary or supplementary DN for an ACD group, or monitor channel)
- CTC/CMP procedures supported
- DN for the device, for example, its telephone number or agent position

You need to use `ctcCmpGetChannelInformation` 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, `ctcCmpGetChannelInformation` returns the following information only:

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

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

Arguments

channel

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

This argument is a `ctcChanId` datatype that contains the channel ID value returned by `ctcCmpAssign` for the device in use.

channelData

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpChanData`. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
ctcCmpChanData {
    unsigned int      lineType;
    unsigned int      procedureSupport;
    unsigned int      attributeSupport;
    ctcDeviceString   setDN;
}
```

The following subsections describe these fields:

- **lineType**

This 32-bit field contains a value that identifies the type of line associated with the assigned device. The following table shows the values that can be returned:

Value	Description
<code>ctcK_LineACD</code>	The channel is assigned to a primary or supplementary DN for an ACD group
<code>ctcK_LineMonitorChannel</code>	The channel is assigned to a monitor channel
<code>ctcK_LineRoutePoint</code>	The channel is assigned to a CDN route point
<code>ctcK_LineVoiceSet</code>	The channel is assigned to an agent position or a telephone

- **procedureSupport**

This 32-bit integer identifies the procedure routines supported by the switch to which the communications channel is assigned.

ctcCmpGetChannelInformation

The following values can be returned in this field (note that not all values can be returned for all device types):

- ctcM_AddMonitor
- ctcM_AnswerCall
- ctcM_Assign
- ctcM_CancelCall
- ctcM_ConferenceJoin
- ctcM_ConsultationCall
- ctcM_Deassign
- ctcM_ErrMsg
- ctcM_GetACDGroupStatus
- ctcM_GetChannelInformation
- ctcM_GetEvent
- ctcM_GetMessageWaiting
- ctcM_GetMonitor
- ctcM_GetRouteQuery
- ctcM_GiveTreatment
- ctcM_HangupCall
- ctcM_HoldCall
- ctcM_MakeCall
- ctcM_ReconnectHeld
- ctcM_RemoveMonitor
- ctcM_RespondToRouteQuery
- ctcM_ResumeCall
- ctcM_SetAgentStatus
- ctcM_SetMonitor
- ctcM_Snapshot
- ctcM_TransferCall

Note that a function-supported mask is not returned for ctcCmpErrMsg. This routine is supported for all channels.

- **attributeSupport**

This 32-bit integer identifies the attribute routines supported by the switch to which the communications channel is assigned. Attribute routines are routines associated with operating modes for the assigned device.

The following values can be returned:

- ctcM_CmpGiveTreatment
- ctcM_GetAcidGroupStatus
- ctcM_GetAgentStatus
- ctcM_GetMessageWaiting

ctcCmpGetChannelInformation

ctcM_GetMonitor
ctcM_GetRoutingEnable
ctcM_SetAgentStatus
ctcM_SetMonitor
ctcM_SetRoutingEnable

- **setDN**

This 24-byte field contains one of the following:

- If the channel is assigned to a device, the device identifying number. The types of device are:
 - Primary DN for an ACD group
 - Supplementary DN for an ACD group
 - Centrex DN
 - Residential DN
 - Secondary DN
 - Agent position
 - CDN
- If the channel is assigned to a monitor channel, a device number generated by CTC/CMP.

You can use this device number with the `ctcCmpAddMonitor` command to receive events for the monitor channel on another monitor channel. 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.

The maximum length for `setDN` is specified by the literal `ctcMaxDnLen`, in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the CTC/CMP API or the CTC/CMP server. Refer to Chapter 3 for details.

ctcCmpGetEvent

ctcCmpGetEvent

Get Information About Event and State Changes

Format in C

```
unsigned int ctcCmpGetEvent
              (ctcChanId          channel,
               ctcCmpEventData    *eventData,
               unsigned int       dontWait)
```

Description

The ctcCmpGetEvent routine returns call and party data associated with the assigned device, or devices monitored on a monitor channel.

It can return details of:

- Call states
- Call events
- Call parties
- 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

The amount of information that CTC/CMP returns depends on the information provided by the DMS-100. This may be different for a call that is local to the DMS-100 and for an outside call, depending on the type of trunks connected to the DMS-100.

Calling ctcCmpGetEvent

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

If you post a ctcCmpGetEvent request and the previous ctcCmpGetEvent request has not yet completed, a ctcEventInProgress error is returned.

Error for Event Data Lost

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

Agent Positions and Supplementary DNs for ACD Groups

To receive events for an agent position or supplementary DN for an ACD group, you use:

1. `ctcCmpAssign` to assign a channel to the primary DN for the ACD group associated with the agent position or supplementary DN.
2. `ctcCmpAssign` to assign a channel to the agent position or supplementary DN for the ACD group.
3. `ctcCmpSetMonitor` to set monitoring on for the agent position or supplementary DN for the ACD group.
4. `ctcCmpGetEvent` to receive events for the agent position or supplementary DN for the ACD group.

Monitor Channels

When you call `ctcCmpGetEvent` for a monitor channel, `ctcCmpGetEvent` returns event information when there is activity on a device monitored on that monitor channel.

You can identify on which device an event has occurred by the information returned in the `monitorParty` field in the `ctcCmpGetEvent` structure. This field returns the DN for the device.

The `nestedMonitorChannel` field identifies whether the device is associated with another monitor channel (that is, currently being monitored on the monitor channel).

Restrictions

`ctcCmpGetEvent` 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 1.11.4).

To receive events for a secondary DN, the associated agent must be logged on.

ctcCmpGetEvent

Arguments

channel

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

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

eventData

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type ctcCmpEventData. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
struct ctcCmpEventData{
    ctcCmpNetCallId  refId;
    unsigned int     state;
    unsigned int     event;
    unsigned int     eventQualifier;
    unsigned int     callMode;
    unsigned int     callType;
    unsigned int     prevApplId;
    ctcNameString    callingPartyName;
    unsigned int     callingPartyType;
    ctcDeviceString  callingParty;
    unsigned int     calledPartyType;
    ctcDeviceString  calledParty;
    ctcDeviceString  dialedParty;
    ctcDeviceString  chargeNumber;
    ctcDeviceString  acdDn;
    ctcDeviceString  acdGroup;
    unsigned int     firstFwdPartyType;
    unsigned int     firstFwdPartyReason;
    ctcDeviceString  firstFwdParty;
    unsigned int     lastFwdPartyType;
    unsigned int     lastFwdPartyReason;
    ctcDeviceString  lastFwdParty;
    ctcCmpApplString applicationData;
    ctcDeviceString  monitorParty;
    ctcDeviceString  nestedMonitorChannel;
    ctcNameString    dialedPartyName;
    ctcTimeStamp     timestamp;
}
```

The strings in this structure are all null-terminated.

The following subsections describe the ctcCmpEventData fields:

- **refId**

This argument contains the address of a fixed-format structure, for which you allocate memory of type ctcCmpNetCallId. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
struct ctcCmpNetCallId {
    unsigned int    netNodeId;
    unsigned int    localCallId;
}
```

The information returned in these fields provides a unique call reference for a particular call. The DMS-100 returns the following information in the fields of the ctcCmpNetCallId structure:

- **netNodeId**

This 32-bit field contains a network node identifier. This is used to identify which DMS-100 first receives a call when a call is handled by more than one DMS-100.

- **localCallId**

This field returns a local call identifier. This is used to uniquely identify the call on the DMS-100.

- **state**

This 32-bit field contains a value that identifies the state of the current call. Table 2-2 shows the possible states of a call, and the corresponding values returned. These call state literals are supplied in a CTC/CMP definitions file (see Section 1.6).

Example state transitions for an outbound call are as follows:

Null → **Active** → **Null**

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

Null → **Receive** → **Active** → **Null**

ctcCmpGetEvent

Table 2–2 Call States Returned by ctcCmpGetEvent

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.
Dialing	ctcK_DialingState	The switch is dialing an outbound call for the assigned 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 is 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	This state is also known as the Idle state. It signals the end of a call.
Queued	ctcK_QueueState	A call has entered the monitored ACD group or a call on the assigned device has been queued.
Receive	ctcK_ReceiveState	The assigned device has received a call, and is ringing (depending on the device).
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.
Unknown	ctcK_UnknownState	The assigned device is in one of the above states.

- **event**

This 32-bit integer identifies the event. Table 2–3 describes the agent event values that can be returned in this field. Table 2–4 describes the call event values that can be returned, and shows the possible call states for each event.

When `ctcCmpGetEvent` returns the information, compare the values returned in the integer with the call event literals supplied in a CTC/CMP definitions file (see Section 1.6).

ctcCmpGetEvent

Table 2–3 Agent Events Returned by ctcCmpGetEvent

Event-Value	Description	ctcCmpEventData Fields Used
ctcK_CmpAgentInformation	The agent has taken additional action during an active call. Check the eventQualifier field for a value that indicates the additional action taken.	eventQualifier callType† calledPartyType† calledParty† acdDn† acdGroup† applicationData‡
ctcK_AgentLoggedOn	The agent has logged in.	calledPartyType calledParty acdDn applicationData
ctcK_AgentLoggedOff	The agent has logged out.	calledPartyType calledParty acdDn
ctcK_AgentMode	The current work mode for an agent. Check the eventQualifier field for a value that indicates the mode.	eventQualifier callMode applicationData
ctcK_AgentModeChange	The work mode for an agent has changed. Check the eventQualifier field for a value that identifies the new work mode.	eventQualifier calledPartyType calledParty acdDn applicationData

† Only used if the value in eventQualifier is ctcK_CmpEMKEvent

‡ Only used if the value in eventQualifier is ctcK_CmpLOBEvent

Table 2–4 Call Events Returned by ctcCmpGetEvent

Event-Value	Description	States
ctcK_CmpDeviceStatus‡	A device status event has been returned on the assigned device (CDNs only).	Null
ctcK_CmpMessageWaiting†	A message waiting indication has been returned on the assigned device.	Null
ctcK_CmpTreatmentComplete†	A RAN or music treatment request has completed on the assigned device.	Queued
ctcK_DestSeized	A call has been successfully offered to an agent position.	Deliver

† NA008 or equivalent

‡ NA009 or equivalent

ctcCmpGetEvent

Table 2–4 Call Events Returned by ctcCmpGetEvent (Continued)

Event-Value	Description	States
ctcK_Error	The call has failed for an unspecified reason. For more information, check the eventQualifier field.	Fail
ctcK_InboundCall	A new call has arrived at the assigned device.	Receive or Queued
ctcK_Offhook†	A new call has been made from the assigned device.	Initiate
ctcK_OpAnswered	The other party answered the call on the assigned device.	Active
ctcK_OpConferenced†	Another party on the call has created a conference call.	Active
ctcK_OpDisconnected	The other party hung up the call.	Active Null
ctcK_TpAnswered	This party has answered an incoming call on the assigned device.	Active
ctcK_TpConferenced†	This party has included another party in a conference call.	Active
ctcK_TpDisconnected	This party has disconnected the call.	Null
ctcK_TpRetrieved†	This party has retrieved a call that was either on hold or in the call-waiting queue.	Active
ctcK_Transferred†	The call has been transferred from another device. This event is returned to both parties on the new call.	Receive or Active

† NA008 or equivalent

- **eventQualifier**

This 32-bit integer provides more detailed information on certain events. Table 2–5 describes the event qualifier values that can be returned in this field.

When ctcCmpGetEvent returns the information, compare the values returned in the integer with the event qualifier literals supplied in a CTC/CMP definitions file (see Section 1.6).

Table 2–5 Event Qualifiers Returned by ctcCmpGetEvent

Qualifier-Value	Description
ctcK_CmpAgentBusy†	The agent is busy with other work.
ctcK_CmpAgentLogout†	The agent has logged out.
ctcK_CmpAgentReady	The agent is ready to take calls.

Table 2–5 Event Qualifiers Returned by ctcCmpGetEvent (Continued)

Qualifier-Value	Description
‡ NA009 or equivalent	
ctcK_CmpAgentNotReady	The agent is not ready to take calls.
ctcK_CmpCallAbandoned	The caller disconnected before the call was answered.
ctcK_CmpCallCallingNameEvent†	A CallCallingName event was received from the switch.
ctcK_CmpCallCleared	The call has been hung up.
ctcK_CmpCallForwarded	The call has been forwarded to another destination.
ctcK_CmpCallOverflowed	An incoming call has been overflowed to another destination for one of the following reasons: <ul style="list-style-type: none"> • The agent is busy • The agent has pressed the Not-Ready key on their telephone set • The agent did not answer within the maximum defined time
ctcK_CmpCallPickedUp	The call has been picked up by another agent after being offered to this party.
ctcK_CmpCallQueuedEvent	A Call-Queued event was received from the switch.
ctcK_CmpCallReceivedEvent	A Call-Received event was received from the switch.
ctcK_CmpCallRedirected	The call has been redirected.
ctcK_CmpCallTransferred	This party has transferred the call and hung up.
ctcK_CmpCDNStatusEvent‡	The device status event is for a CDN route point.
ctcK_CmpEMKEvent‡	The agent has activated or terminated the Emergency Key feature during an active call.
ctcK_CmpLOBEvent‡	The agent has entered a line of business (LOB) code during an active call.
ctcK_CmpPartyDropped	The other party or the consultation party in a three-way call has left the call.
† NA008 or equivalent	
‡ NA009 or equivalent	

ctcCmpGetEvent

- **callMode**

This 32-bit field indicates the call mode. It can contain one of the following values:

Value	Description
ctcK_ActiveState‡	The agent's secondary DN is in the active state (for ctcK_AgentMode events only).
ctcK_CallWaitied	The call is waiting at the assigned device because the device is currently busy (for ctcK_DestSeized, ctcK_InboundCall, ctcK_OpAnswered, and ctcK_TpAnswered events only).
ctcK_CmpControlled‡	An incoming call to a CDN route point is redirected to the CTC/CMP server (for ctcK_CmpDeviceStatus events only).
ctcK_CmpDefault‡	An incoming call to a CDN route point is redirected to the default ACD DN (for ctcK_CmpDeviceStatus events only).
ctcK_CmpRevertToDefault‡	Incoming and currently queued calls are redirected from a CDN route point to the default ACD DN (for ctcK_CmpDeviceStatus events only).
ctcK_ReceiveState‡	The agent's secondary DN is in the receiving state (for ctcK_AgentMode events only).
ctcK_UnavailableState‡	The agent's secondary DN is in the unavailable state (for ctcK_AgentMode events only).

‡ NA009 or equivalent

- **callType**

This 32-bit integer identifies the type of inbound call.

If the event is ctcK_InboundCall, this integer contains one of the following values:

Value	Description
ctcK_CallNewCall	The call has not yet been forwarded.
ctcK_CallForwarded	The call has been forwarded from another destination.
ctcK_CallOverflowed	The call has been overflowed from another destination.
ctcK_CallRedirected	The call has been redirected.
ctcK_CallTransferred	The call has been transferred.

ctcCmpGetEvent

If the event is `ctcK_CmpAgentInformation`, with a `ctcK_CmpEMKEvent` event qualifier, this integer contains one of the following values:

Value	Description
<code>ctcK_CmpEMKActivated</code>	The agent has activated the emergency key.
<code>ctcK_CmpEMKAgentCancels</code>	The agent has cancelled the emergency key by re-pressing the key.
<code>ctcK_CmpEMKAgentExits</code>	The agent has exited the call.
<code>ctcK_CmpEMKAuxExits</code>	The auxiliary has exited the call.
<code>ctcK_CmpEMKAuxTimeout</code>	The emergency key is deactivated by auxiliary timeout.
<code>ctcK_CmpEMKCallerExits</code>	The caller has exited the call.
<code>ctcK_CmpEMKDeactivated</code>	The agent has deactivated the emergency key by pressing the 'not ready' key.
<code>ctcK_CmpEMKSuprExits</code>	The supervisor has exited the call.
<code>ctcK_CmpEMKSuprTimeout</code>	The emergency key is deactivated by supervisor timeout.

- **prevApplId**

This 32-bit field contains the identifier you set when configuring the link between the CTC/CMP server and the DMS-100. The identifier uniquely identifies the communications link between the CTC/CMP server and the switch. Refer to the *CT-Connect for CompuCALL Installation and Administration Guide* for more information about setting link configuration parameters.

- **callingPartyName**

This character string returns the name of the calling party, if available. This field is only applicable for Centrex and residential lines.

The maximum length for `callingPartyName` is specified by the literal `ctcCmpNameLen`. Note that this maximum length includes the null termination character (NUL).

ctcCmpGetEvent

- **callingPartyType**

This 32-bit field identifies the type of number associated with the calling party. It can contain one of the following values:

Value	Description
ctcK_NumUnknown	The type of number is unknown.
ctcK_NumInternat	The number is an international directory number.
ctcK_NumNational	The number is a national directory number.
ctcK_NumSubscriber	The number is a subscriber number, for example, an extension number.

- **callingParty**

This character string contains the number of the calling party.

The maximum length for callingParty is specified by the literal ctcMaxDnLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **calledPartyType**

This 32-bit field identifies the type of number associated with the called party.

It can contain one of the following values:

Value	Description
ctcK_Dn	Identifies the number as a Directory Number (DN). For example, an extension number.
ctcK_AgentPosition	Identifies the number as a position identifier for an agent position.

- **calledParty**

This character string contains:

- For call events, the number for the destination party.
- For agent events, the position ID for the agent.
- For a ctcK_CmpAgentInformation agent event, where the event qualifier is ctcK_CmpEMKEvent, the position ID for the agent supervisor.

The maximum length for calledParty is specified by the literal ctcMaxDnLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **dialedParty**

This character string contains the DN for the dialed party. The dialed party is the party originally called. For example, if A calls B, and then B transfers the call to C (the monitored device), the dialed party is B.

The maximum length for dialedParty is specified by the literal `ctcMaxDnLen` in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **chargeNumber**

If available, this character string contains the Automatic Number Identification (ANI) charge number.

The maximum length for chargeNumber is specified by the literal `ctcMaxDnLen` in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **acdDn**

This character string contains:

- The primary or supplementary DN for the ACD group handling the call.
- For a `ctcK_CmpAgentInformation` event, with a `ctcK_CmpEMKEvent` event qualifier, the supervisor or auxiliary ACD DN.

The maximum length for acdDn is specified by the literal `ctcMaxDnLen` in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **acdGroup**

This character string contains:

- The primary DN for the ACD group handling the call.
- For a `ctcK_CmpAgentInformation` event, with a `ctcK_CmpEMKEvent` event qualifier, the ACD DN of the agent.

The maximum length for acdGroup is specified by the literal `ctcMaxDnLen` in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

ctcCmpGetEvent

- **firstFwdPartyType**

This 32-bit field identifies the type of number associated with the first forwarding party, when a call has been forwarded by more than one party. It can contain one of the following values:

Value	Description
ctcK_NumUnknown	The type of number is unknown.
ctcK_NumInternat	The number is an international directory number.
ctcK_NumNational	The number is a national directory number.
ctcK_NumSubscriber	The number is a subscriber number, for example, an extension number.

- **firstFwdPartyReason**

This 32-bit field identifies the reason a call was forwarded by the first forwarding party. It can contain one of the following values:

Value	Description
ctcK_Unknown	The forwarding reason is not known or is unavailable.
ctcK_UserBusy	The call was forwarded because the first party was busy.
ctcK_NoReply	The first party did not answer the call within a specified time limit.
ctcK_Unconditional	The first party has forwarded all calls unconditionally.

- **firstFwdParty**

This character string contains the number for the first party.

The maximum length for firstFwdParty is specified by the literal ctcMaxDnLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **lastFwdPartyType**

This 32-bit field identifies the type of number associated with the last forwarding party, when a call has been forwarded by more than one party. It can contain one of the following values:

Value	Description
ctcK_NumUnknown	The type of number is unknown.
ctcK_NumInternat	The number is an international directory number.
ctcK_NumNational	The number is a national directory number.
ctcK_NumSubscribe	The number is a subscriber number, for example, an extension number.

- **lastFwdPartyReason**

This 32-bit field identifies the reason a call was forwarded by the last forwarding party. It can contain one of the following values:

Value	Description
ctcK_Unknown	The forwarding reason is not known or is unavailable.
ctcK_UserBusy	The call was forwarded because the last party was busy.
ctcK_NoReply	The last party did not answer the call within a specified time limit.
ctcK_Unconditional	The last party has forwarded all calls unconditionally.

- **lastFwdParty**

This character string contains the number for the last forwarding party.

The maximum length for lastFwdParty is specified by the literal ctcMaxDnLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **applicationData**

This field returns data that has been associated with a call and stored by the switch. The data is returned as a character string.

After some events, the data in this field has a specific meaning, as follows:

- For a ctcK_CmpAgentInformation event, with a ctcK_CmpLOBEvent event qualifier, this field contains the LOB code.
- For a ctcK_AgentLoggedOn event, this field contains the agent identifier.

ctcCmpGetEvent

- For a ctcK_AgentChangeMode or ctcK_AgentMode event, with a ctcK_AgentNotReady event qualifier, the field contains the walkaway reason.

The maximum length for applicationData is specified by the literal ctcCmpAppDnLen in a CTC/CMP definitions file (see Section 1.6). 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 ctcCmpAssign 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/CMP definitions file (see Section 1.6). 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/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **dialedPartyName**

This character string returns the name of the dialed party, if available. This field is only applicable for Centrex and residential lines.

The maximum length for dialedPartyName is specified by the literal ctcCmpNameLen. Note that this maximum length includes the null termination character (NUL).

- **timeStamp**

This field contains the date and time the CTC/CMP server received the event.

It contains a fixed-format structure:

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

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.

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.

ctcCmpGetEvent

Note that if you are writing an application for a large call center (over 50 agents), continuously polling for events will generate a significant amount of network traffic. To avoid network and system resource problems, Dialogic recommends that you do not set this value if your application is intended for this type of environment. If you do use polling, you should leave at least a 500 ms delay between polls.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the CTC/CMP API or the CTC/CMP server.

The CTC/CMP server can return the following event-related condition values:

- `ctcEventDataLost`
- `ctcEventInProgress`
- `ctcInvChannel`
- `ctcLinkDown`
- `ctcMonitorOff`

For details of other condition values that can be returned by the CTC/CMP API or CTC/CMP server for this routine, refer to Chapter 3.

ctcCmpGetMessageWaiting

Get Information About the Message Waiting Indicator

Format in C

```
unsigned int ctcCmpGetMessageWaiting
            (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 ctcCmpAssign 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 contains one of the values in the following table:

Value	Description
ctcK_CmpActivated	Message waiting indicator is activate That is, a voice mail message is waiting.
ctcK_CmpDeactivated	Message waiting indicator deactivated. That is, all messages have been retrieved.
ctcK_CmpExecActivated	As ctcK_CmpActivated, except for collection method.
ctcK_CmpExecDeactivated	As ctcK_CmpDeactivated, except for collection method.
ctcK_CmpGMWNotAllowed	This DN does not provide any message waiting modes.

ctcCmpGetMessageWaiting

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following condition values for this routine:

- `ctcInvChannel`
- `ctcInvDN`
- `ctcInvParam`
- `ctcNotAllowed`

For details of the condition values that can be returned by the CTC/CMP API and CTC/CMP server, refer to Chapter 3.

ctcCmpGetMonitor

Get Information About the Monitoring State

Format in C

```

unsigned int ctcCmpGetMonitor
            (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 ctcCmpSetMonitor routine.

Restrictions

This routine is not supported for channels assigned to monitor channels.

Arguments

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

This argument is a ctcChanId datatype that contains the channel ID value returned by ctcCmpAssign 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

ctcCmpGetMonitor

Condition Values Returned

When the routine completes successfully, *CTC/CMP* returns the condition value `ctcSuccess`. If the routine does not complete successfully, *CTC/CMP* returns a condition value generated by the *CTC/CMP* API or the *CTC/CMP* server. Refer to Chapter 3 for details.

ctcCmpGetRouteQuery

Get Route Query Messages from the Switch

Format in C

```

unsigned int ctcCmpGetRouteQuery
              (ctcChanId          channel,
               ctcCmpRouteData   *routeData,
               unsigned int      dontWait)

```

Description

The `ctcCmpGetRouteQuery` routine presents call route request from the switch so that your application can provide new destinations for routed calls. This routine returns the following information (if provided by the switch):

- Call type
- Calling party
- Dialed number
- Charge number
- Forwarding parties
- Application data passed with the call

Call Routing

Call routing operates differently for primary ACD DN and CDN route points, as follows:

- Primary ACD DN

Call routing allows your application to redirect incoming calls for a primary DN for an ACD group to another destination. Your application must assign a channel to the primary DN for an ACD group. Whenever a CALL-REDIRECT request comes in on the primary DN, your application will receive a route request.

When your application receives a route request, use `ctcCmpGetRouteQuery` to return data relating to the call, including the call reference identifier and route identifier. Your application can then define a new destination for the call by entering the call reference identifier and the route identifier as parameters to a `ctcCmpRespondToRouteQuery` routine. If the new destination is valid, the call is redirected.

ctcCmpGetRouteQuery

- **CDN Route point**

Call routing allows your application to route queued calls for a CDN to another destination. Your application must assign a channel to the CDN route point. Whenever a CALL-QUEUED event comes in on the CDN, your application will receive a route request.

When your application receives a route request, use `ctcCmpGetRouteQuery` to return data relating to the call, including the call reference identifier. Your application can give treatment to the call by using the `ctcCmpGiveTreatment` routine before routing to a new destination. When defining a new destination for the call, your application must pass the call reference identifier as a parameter to a `ctcCmpRespondToRouteQuery` routine. If the new destination is valid, the call is redirected.

Please note that your application can only route a call to a CDN if the CDN is in CONTROLLED mode (for details, see the description of the `ctcCmpSetRoutingEnable` routine).

If you are also monitoring these DNs, a `ctcK_InboundCall` event with a `ctcK_QueueState` state is received for the primary ACD DN or CDN. The event qualifier will indicate the CompuCALL message type which resulted in the event as follows:

Event Qualifier	CompuCALL Message Type
<code>ctcK_CmpCallReceivedEvent</code>	CALL-RECEIVED
<code>ctcK_CmpCallQueuedEvent</code>	CALL-QUEUED

Choosing Not to Reroute a Call

If you do not want the application to reroute a call presented by `ctcCmpGetRouteQuery`, post a call to the `ctcCmpRespondToRouteQuery` 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. `ctcCmpAssign` to assign the channel.
For CDNs, you must also use `ctcCmpSetRoutingEnable` and set `routingMode` to `ctcK_CmpControlled`.
2. `ctcCmpGetRouteQuery` to be notified of route requests for calls to the DN.
3. `ctcCmpRespondToRouteQuery` to supply a route for the call.

ctcCmpGetRouteQuery

Note that if you post a `ctcCmpGetRouteQuery` request before the previous `ctcCmpGetRouteQuery` request has completed, a `ctcRouteInProgress` error is returned.

Restrictions

The following restrictions apply to this routine:

- `ctcCmpGetRouteQuery` is not supported for channels assigned to telephones, supplementary ACD DNs, agent positions, or monitor channels.
- `ctcCmpGetRouteQuery` returns data relating to a call 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 1.11.4) while waiting for the route request.

Arguments

channel

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

This argument is a `ctcChanId` datatype that contains the channel ID value returned by `ctcCmpAssign` for the device in use.

routeData

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpRouteData`. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
struct ctcCmpRouteData{
    ctcCmpNetCallId  callRefId;
    unsigned int    routeId;
    unsigned int    callType;
    unsigned int    prevApplId;
    unsigned int    callingPartyType;
    ctcDeviceString callingParty;
    ctcDeviceString dialedParty;
    ctcDeviceString chargeNumber;
    ctcDeviceString acdDn;
    ctcDeviceString acdGroup;
    unsigned int    firstFwdPartyType;
    unsigned int    firstFwdPartyReason;
    ctcDeviceString firstFwdParty;
```

ctcCmpGetRouteQuery

```
    unsigned int    lastFwdPartyType;  
    unsigned int    lastFwdPartyReason;  
    ctcDeviceString lastFwdParty;  
    ctcCmpApplString applicationData;  
    ctcTimeStamp    timeStamp;  
};
```

CTC/CMP returns information in one or more of these fields.

The following sections describe the ctcCmpRouteData fields.

- **callRefId**

This field contains the address of a fixed-format structure, for which you allocate memory of type ctcCmpNetCallId. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
struct ctcCmpNetCallId {  
    unsigned int    netNodeId;  
    unsigned int    localCallId;  
}
```

The information returned in these fields provides a unique call reference for a particular call.

The DMS-100 returns an identifier in each field:

- **netNodeId**

This 32-bit field returns a network node identifier. This is used to identify which DMS-100 first receives a call when a call is handled by more than one DMS-100.

- **localCallId**

This field returns a local call identifier. This is used to uniquely identify the call on the DMS-100.

- **routeId**

This 32-bit field contains the route identifier value to be passed as the routeId argument for the ctcCmpRespondToRouteQuery routine.

This value is zero if the route request was generated for a CDN route point from the CompuCALL Call-Queued message.

- **callType**

This 32-bit integer identifies the type of route request. It can contain one of the following values:

Value	Description
ctcK_CallNewCall	The call has not yet been forwarded.
ctcK_CallForwarded	The call has been forwarded from another destination.
ctcK_CallOverflowed	The call has been overflowed from another destination.
ctcK_CallRedirected	The call has been redirected.
ctcK_CallReturnedToQueue	The call has been returned to the incoming call queue.
ctcK_CallTransferred	The call has been transferred.

When ctcCmpGetEvent returns the information, compare the values returned in the integer with the event qualifier literals supplied in a CTC/CMP definitions file (see Section 1.6).

- **prevApplId**

This 32-bit field contains the identifier you set when configuring the link between the CTC/CMP server and the DMS-100. The identifier uniquely identifies the communications link between the CTC/CMP server and the switch. Refer to the *CT-Connect for CompuCALL Installation and Administration Guide* for more information about setting link configuration parameters.

- **callingPartyType**

This 32-bit field identifies the type of number associated with the calling party. It can contain one of the following values:

Value	Description
ctcK_NumUnknown	The type of number is unknown.
ctcK_NumInternat	The number is an international directory number.
ctcK_NumNational	The number is a national directory number.
ctcK_NumSubscriber	The number is a subscriber number, for example, an extension number.

ctcCmpGetRouteQuery

- **callingParty**

This character string contains the number of the calling party.

The maximum length for callingParty is specified by the literal ctcMaxDnLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **dialedParty**

This character string contains the DN for the dialed party. The dialed party is the party originally called. For example, if A calls B, and then B transfers the call to C (the monitored device), the dialed party is B.

The maximum length for dialedParty is specified by the literal ctcMaxDnLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **chargeNumber**

If available, this character string contains the Automatic Number Identification (ANI) charge number if available from the DMS-100.

The maximum length for chargeNumber is specified by the literal ctcMaxDnLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **acdDn**

This character string contains the primary or supplementary DN for the ACD group handling the call.

The maximum length for acdDn is specified by the literal ctcMaxDnLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **acdGroup**

This character string contains the primary DN for the ACD group handling the call.

The maximum length for acdGroup is specified by the literal ctcMaxDnLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **firstFwdPartyType**

This 32-bit field identifies the type of number associated with the first forwarding party, when a call has been forwarded by more than one party. It can contain one of the following values:

Value	Description
ctcK_NumUnknown	The type of number is unknown.
ctcK_NumInternat	The number is an international directory number.
ctcK_NumNational	The number is a national directory number.
ctcK_NumSubscribe	The number is a subscriber number, for example, an extension number.

- **firstFwdPartyReason**

This 32-bit field identifies the reason a call was forwarded by the first forwarding party. It can contain one of the following values:

Value	Description
ctcK_Unknown	The forwarding reason is not known or is unavailable.
ctcK_UserBusy	The call was forwarded because the first party was busy.
ctcK_NoReply	The first party did not answer the call within a specified time limit.
ctcK_Unconditional	The first party has forwarded all calls unconditionally.

- **firstFwdParty**

This character string contains the number for the first forwarding party.

The maximum length for firstFwdParty is specified by the literal ctcMaxDnLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

ctcCmpGetRouteQuery

- **lastFwdPartyType**

This 32-bit field identifies the type of number associated with the last forwarding party, when a call has been forwarded by more than one party. It can contain one of the following values:

Value	Description
ctcK_NumUnknown	The type of number is unknown.
ctcK_NumInternat	The number is an international directory number.
ctcK_NumNational	The number is a national directory number.
ctcK_NumSubscribe	The number is a subscriber number, for example, an extension number.

- **lastFwdPartyReason**

This 32-bit field identifies the reason a call was forwarded by the last forwarding party. It can contain one of the following values:

Value	Description
ctcK_Unknown	The forwarding reason is not known or is unavailable.
ctcK_UserBusy	The call was forwarded because the last party was busy.
ctcK_NoReply	The last party did not answer the call within a specified time limit.
ctcK_Unconditional	The last party has forwarded all calls unconditionally.

- **lastFwdParty**

This character string contains the number for the last forwarding party.

The maximum length for lastFwdParty is specified by the literal ctcMaxDnLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **applicationData**

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

The maximum length for applicationData is specified by the literal ctcAppDnLen in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

- **timeStamp**

This field contains the date and time the CTC/CMP server received the event.

It contains a fixed-format structure:

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

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.

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.

ctcCmpGetRouteQuery

Note that if you are writing an application for a large call center (over 50 agents), continuously polling for events will generate a significant amount of network traffic. To avoid network and system resource problems, Dialogic recommends that you do not set this value if your application is intended for this type of environment. If you do use polling, you should leave at least a 500 ms delay between polls.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the CTC/CMP API or the CTC/CMP server.

The CTC/CMP server can return the following routine-related condition values:

- `ctcInvChannel`
- `ctcLinkDown`
- `ctcMonitorOff`
- `ctcRouteDataLost`
- `ctcRouteInProgress`

For details of other condition values that can be returned by the CTC/CMP API or CTC/CMP server for this routine, refer to Chapter 3.

ctcCmpGetRoutingEnable

Get Information About the Routing State

Format in C

```
unsigned int ctcCmpGetRoutingEnable (ctcChanId channel)
```

Description

The `ctcCmpGetRoutingEnable` routine gets the routing status of the assigned CDN route point. The status is returned by the DMS-100 through a `ctcK_CmpDeviceStatus` event.

To enable or disable routing for a CDN route point, use the `ctcCmpSetRoutingEnable` routine.

Restrictions

The following restrictions apply:

- This routine requires NA009 (or equivalent) or later.
- This routine is only supported for CDN route points.

Arguments

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

This argument is a `ctcChanId` datatype that contains the channel ID value returned by `ctcCmpAssign` for the CDN route point in use.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the CTC/CMP API or the CTC/CMP server. For details, refer to Chapter 3.

ctcCmpGiveTreatment

ctcCmpGiveTreatment Give Treatment to a Message

Format in C

```
unsigned int ctcCmpGiveTreatment
            (ctcChanId      channel,
             ctcCmpNetCallId *callRefId,
             unsigned int   toneType,
             unsigned int   announcementRoute,
             ctcCmpApplString announcementName)
```

Description

This routine directs the DMS-100 to apply a treatment to a call which is queued to a CDN route point .

Restrictions

This routine is only supported for CDN route points.

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 ctcCmpAssign for the device in use.

callRefId

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type ctcCmpNetCallId. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
ctcCmpNetCallId {
    unsigned int   netNodeId;
    unsigned int   localCallId;
};
```

ctcCmpGiveTreatment

Information in the ctcCmpNetCallId structure provides a unique call reference identifier for the call to which treatment is applied:

- **netNodeId**

This 32-bit field contains a network node identifier. This is used to identify which DMS-100 first receives a call when the call is handled by more than one DMS-100.

- **localCallId**

This field returns a local call identifier. This is used to uniquely identify the call on the DMS-100.

toneType

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_CmpAnnouncement	Call remains in the queue. An announcement is applied to the call. Specify the announcementRoute and announcementName arguments.
ctcK_CmpBusy	Call is removed from the queue and routed to Busy.
ctcK_CmpDisconnect	Call is removed from the queue and cleared.
ctcK_CmpFastBusy	Call is removed from the queue and routed to Fastbusy.
ctcK_CmpRingBack	Call remains in the queue. Ringback treatment is applied to the call. Treatments on a queued call can be interrupted.
ctcK_CmpSilence	Call remains in the queue. Silence treatment is applied to the call. Treatments on a queued call can be interrupted.

announcementRoute

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

This argument is a 32-bit integer that specifies which announcement to play. You can specify a value in the range 0 to 512.

ctcCmpGiveTreatment

announcementName

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

This argument is the address of a character string that indicates the audio type that you want the call to be treated with. The audio types are RAN, MUSIC, or AUDIO.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following condition values for this routine:

- `ctcBadObjState`
- `ctcInvParam`
- `ctcMissingParam`
- `ctcNotAllowed`
- `ctcUnavailable`

For details of the condition values that can be returned by the CTC/CMP API and CTC/CMP server, refer to Chapter 3.

ctcCmpHangupCall

Disconnect a Call

Format in C

```
unsigned int ctcCmpHangupCall (ctcChanId channel,  
                                ctcCmpNetCallId *callRefId)
```

Description

The `ctcCmpHangupCall` 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 `ctcCmpHangupCall`, the switch will re-call you. To end a consultation call and reconnect to the original party, use `ctcCmpReconnectHeld`.

The call reference identifier information for the hangup call is returned by the DMS-100.

Restrictions

This routine is not supported for channels assigned to ACD groups, CDNs, or monitor channels.

After a `ctcCmpHangupCall`, a user at a 500/2500 set receives the dial tone. Further calls can then be made. However, if the user stays off-hook for too long, the set will receive a lockout treatment.

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 `ctcCmpAssign` for the device in use.

ctcCmpHangupCall

callRefId

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpNetCallId`. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
ctcCmpNetCallId {  
    unsigned int    netNodeId;  
    unsigned int    localCallId;  
};
```

Information in the `ctcCmpNetCallId` structure provides a unique call reference identifier for the conference call. The DMS-100 returns the following information in the `ctcCmpNetCallId` structure:

- **netNodeId**

This 32-bit field contains a network node identifier. This is used to identify which DMS-100 first receives a call when the call is handled by more than one DMS-100.

- **localCallId**

This field returns a local call identifier. This is used to uniquely identify the call on the DMS-100.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following condition values for this routine:

```
ctcBadObjState  
ctcInvParam  
ctcMissingParam  
ctcNotAllowed  
ctcUnavailable
```

For details of the condition values that can be returned by the CTC/CMP API and CTC/CMP server, refer to Chapter 3.

ctcCmpHoldCall

Put Current Call on Hold

Format in C

```
unsigned int ctcCmpHoldCall (ctcChanId      channel,
                             ctcCmpNetCallId *callRefId)
```

Description

The `ctcCmpHoldCall` routine puts the current call on the assigned device on a hard hold. To retrieve this held call, use the `ctcCmpResumeCall` routine.

The call reference identifier information for the call is returned by the DMS-100.

Restrictions

This routine is not supported for channels assigned to ACD groups, CDNs, or monitor channels. In addition, it is only available for certain types of key or business set.

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 `ctcCmpAssign` for the device in use.

callRefId
 type: **ctcCmpNetCallId**
 access: **write only**
 mechanism: **by reference**

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpNetCallId`. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
ctcCmpNetCallId {
    unsigned int    netNodeId;
    unsigned int    localCallId;
};
```

ctcCmpHoldCall

Information in the `ctcCmpNetCallId` structure provides a unique call reference identifier for the conference call. The DMS-100 returns the following information in the `ctcCmpNetCallId` structure:

- **netNodeId**

This 32-bit field contains a network node identifier. This is used to identify which DMS-100 first receives a call when the call is handled by more than one DMS-100.

- **localCallId**

This field returns a local call identifier. This is used to uniquely identify the call on the DMS-100.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following condition values for this routine:

- `ctcBadObjState`
- `ctcInvParam`
- `ctcMissingParam`
- `ctcNotAllowed`
- `ctcUnavailable`

For details of the condition values that can be returned by the CTC/CMP API and CTC/CMP server, refer to Chapter 3.

ctcCmpMakeCall

Make a Call

Format in C

```

unsigned int ctcCmpMakeCall
    (ctcChanId          channel,
     ctcDeviceString   calledNumber,
     unsigned int      makeCallType,
     ctcCmpCodeString authCodeDigits,
     ctcCmpCodeString acctCodeDigits,
     ctcCmpNetCallId  *callRefId)

```

Description

The `ctcCmpMakeCall` routine makes a call from an agent position or a telephone set to any number that the DMS-100 recognizes as valid.

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

The call reference identifier information for the call is returned by the DMS-100.

On-hook Dialing

If the telephone is a feature phone with a loudspeaker, the switch can provide on-hook dialing. This means that the agent can initiate a call and place a call without lifting the handset (for more details, refer to the description of the `makeCallType` argument).

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 rings or buzzes the originating telephone.
3. The switch or user sets the telephone off-hook, and the switch sets up the connection to the destination.

Restrictions

This routine is not supported for channels assigned to ACD groups, CDNs, or monitor channels.

ctcCmpMakeCall

Arguments

channel

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

This argument is a `ctcChanId` datatype that contains the channel ID value returned by `ctcCmpAssign` 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/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

makeCallType

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

This 32-bit integer identifies the specific agent state required before the call proceeds. Specify one of the following values:

Value	Description
<code>ctcK_AgentBeepHset</code>	The agent must be logged in and ready. A tone is applied to the telephone set and the outgoing call is placed without the agent going off-hook.
<code>ctcK_AgentBuzzBase</code>	The agent must be logged in and ready. A buzz is applied to the telephone set and the outgoing call is placed without the agent going off-hook.
<code>ctcK_AgentOnline†‡</code>	The agent must be logged in and ready (logically on-hook and idle).
<code>ctcK_AgentReserved†</code>	The agent is not ready to receive inbound calls and is available for outbound calls only (the ACDNR key on the agent set is activated).
<code>ctcK_AgentNotReserved†</code>	The agent is ready to receive both inbound and outbound calls (the ACDNR key on the agent set must not be activated).

† See note following this table

‡ This is the only value valid with secondary DN, Centrex, or residential telephones

ctcCmpMakeCall

Note that with the `ctcK_AgentOnline`, `ctcK_AgentReserved`, and `ctcK_AgentNotReserved` states, the outbound call is placed when the agent goes off-hook manually (or issues a `ctcCmpAnswerCall`). If the agent goes off-hook after ten seconds and the ring threshold time is greater than ten seconds, the `ctcK_Timeout` error will be returned even though the call may be placed successfully. However, if Call Forcing is in effect, the agent will be presented with the outbound call after a short tone burst (that is, the agent does not have to go off-hook).

authCodeDigits

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

This argument is the address of a character string that contains one of the following:

- Authorization code digits that allow an agent to access a specific Network Class Of Service (NCOS). These digits should be specified if the agent does not want to access the default NCOS as defined on the switch.
- Authorization code digits that have been categorized into authorization code types as a security measure on the switch.

The maximum length for `authCodeDigits` is specified by the literal `ctcCodeLen` in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

If the authorization codes are not used on your switch, enter the address of a zero-length character string.

acctCodeDigits

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

This argument is the address of a character string that contains an account code, if defined on the switch. For example, for cost allocation purposes internal to the switch.

Note that it is possible that the authorization code and account code are combined on the DMS-100. If this is the case, the account code must be passed with the authorization code using the `authCodeDigits` argument.

The maximum length for `acctCodeDigits` is specified by the literal `ctcCodeLen` in a CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

ctcCmpMakeCall

Passing data with this argument is optional. If an account code is not defined on the DMS-100 specify the address of a zero-length character string.

callRefId

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpNetCallId`. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
struct ctcCmpNetCallId {
    unsigned int    netNodeId;
    unsigned int    localCallId;
}
```

Information in the `ctcCmpNetCallId` structure provides a unique call reference for the call. The DMS-100 returns the following information in the fields in the `ctcCmpNetCallId` structure:

- **netNodeId**

This 32-bit field returns a network node identifier. This is used to identify which DMS-100 first receives a call when a call is handled by more than one DMS-100.

- **localCallId**

This field returns a local call identifier. This is used to uniquely identify the call on the DMS-100.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following routine-related condition values:

```
ctcBadObjState
ctcCallAborted
ctcInvalidDest
ctcInvCallingDevice
ctcInvCallType
ctcInvCode
ctcMissingParam
```

ctcCmpMakeCall

ctcNotAllowed
ctcNotLoggedIn
ctcNotSubscribed
ctcOrigTimeout
ctcUnavailable

For details of the condition values that can be returned by the CTC/CMP API or CTC/CMP server for this routine, refer to Chapter 3.

ctcCmpReconnectHeld

ctcCmpReconnectHeld Reconnect a Call on Hold

Format in C

```
unsigned int ctcCmpReconnectHeld
            (ctcChanId          channel,
             ctcCmpNetCallId   *callRefId)
```

Description

The `ctcCmpReconnectHeld` routine disconnects a consultation call and reconnects a held call.

The call reference identifier information for the call is returned by the DMS-100.

The following example sequence shows how to use `ctcCmpReconnectHeld`:

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

Restrictions

This routine is not supported for channels assigned to ACD groups, or monitor channels.

Arguments

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

This argument is a `ctcChanId` datatype that contains the channel ID value returned by `ctcCmpAssign` for the device in use.

ctcCmpReconnectHeld

callRefId

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpNetCallId`. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
struct ctcCmpNetCallId {  
    unsigned int    netNodeId;  
    unsigned int    localCallId;  
}
```

Information in the `ctcCmpNetCallId` structure provides a unique call reference for the call. The DMS-100 returns the following information in the fields in the `ctcCmpNetCallId` structure:

- **netNodeId**

This 32-bit field returns a network node identifier. This is used to identify which DMS-100 first receives a call when a call is handled by more than one DMS-100.

- **localCallId**

This field returns a local call identifier. This is used to uniquely identify the call on the DMS-100.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following routine-related condition values:

```
ctcBadObjState  
ctcInvParam  
ctcMissingParam  
ctcNotAllowed  
ctcUnavailable
```

For details of the condition values that can be returned by the CTC/CMP API or CTC/CMP server for this routine, refer to Chapter 3.

ctcCmpRemoveMonitor

ctcCmpRemoveMonitor Removes a Device From a Monitor Channel

Format in C

```
unsigned int ctcCmpRemoveMonitor  
           (ctcChanId    channel,  
            ctcDeviceString deviceDN)
```

Description

The `ctcCmpRemoveMonitor` 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 `ctcCmpDeassign`.

Restrictions

This routine is only supported for channels assigned to monitor channels.

Arguments

channel

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

This argument is a `ctcChanId` datatype that contains the channel ID value returned by `ctcCmpAssign` 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 #.

ctcCmpRemoveMonitor

- For a monitor channel, specify the device number returned in the setDN field of the ctcChanData structure. This is returned when you call ctcCmpGetChannelInformation for the monitor channel. See the description of ctcCmpGetChannelInformation 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/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value ctcSuccess. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the CTC/CMP API or the CTC/CMP server. For details, refer to Chapter 3.

ctcCmpRespondToRouteQuery

ctcCmpRespondToRouteQuery

Respond to Route Query Messages from the Switch

Format in C

```
unsigned int ctcCmpRespondToRouteQuery
            (ctcChanId      channel,
             unsigned int   routId,
             ctcCmpNetCallId *callRefId,
             unsigned int   numberType,
             ctcDeviceString newCalledNumber,
             ctcCmpAppString applicationData)
```

Description

The `ctcCmpRespondToRouteQuery` routine supplies a new route for a call that was presented for routing by `ctcCmpGetRouteQuery`.

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

Restrictions

This routine is not supported for channels assigned to telephones, supplementary DNs, agent positions, or monitor channels.

Arguments

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

This argument is a `ctcChanId` datatype that contains the channel ID value returned by `ctcCmpAssign` for the device in use.

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

This argument is the route ID as returned by the `ctcCmpGetRouteQuery` routine.

ctcCmpRespondToRouteQuery

callRefId

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpNetCallId`. It contains the `callRefId` for the call to be routed as returned by `ctcCmpGetRouteQuery`. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
struct ctcCmpNetCallId {  
    unsigned int    netNodeId;  
    unsigned int    localCallId;  
}
```

Information in the `ctcCmpNetCallId` structure provides a unique call reference identifier for the call.

numberType

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

This argument is the number type for the `newCalledNumber`, as follows:

Value	Description
<code>ctcK_AgentPosition</code>	An agent position ID (for CDNs only)
<code>ctcK_DN</code>	A DN within or outside the DMS-100 (in dialed digits format)

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/CMP definitions file (see Section 1.6). 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.

ctcCmpRespondToRouteQuery

applicationData

type: **ctcCmpApplString**

access: **read only**

mechanism: **by reference**

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.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following condition values for this routine:

- `ctcNotAllowed`
- `ctcInvalidDest`
- `ctcMissingParam`
- `ctcInvParam`

For details of the condition values that can be returned by the CTC/CMP API or CTC/CMP server for this routine, refer to Chapter 3.

ctcCmpResumeCall

Resume a Call on Hold

Format in C

```

unsigned int ctcCmpResumeCall
              (ctcChanId      channel,
               ctcCmpNetCallId *callRefId)

```

Description

The `ctcCmpResumeCall` routine resumes a call that was placed on hold using the `ctcCmpHoldCall` routine.

The call reference identifier information for the call is returned by the DMS-100.

Restrictions

This routine is not supported for channels assigned to ACD groups, CDNs, or 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 `ctcCmpAssign` for the device in use.

The `ctcChanId` datatype is defined in a CTC/CMP definitions file (see Section 1.6).

callRefId
 type: **ctcCmpNetCallId**
 access: **write only**
 mechanism: **by reference**

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpNetCallId`.

ctcCmpResumeCall

The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
ctcCmpNetCallId {
    unsigned int    netNodeId;
    unsigned int    localCallId;
};
```

Information in the `ctcCmpNetCallId` structure provides a unique call reference identifier for the conference call. The DMS-100 returns the following information in the `ctcCmpNetCallId` structure:

- **netNodeId**

This 32-bit field contains a network node identifier. This is used to identify which DMS-100 first receives a call when the call is handled by more than one DMS-100.

- **localCallId**

This field returns a local call identifier. This is used to uniquely identify the call on the DMS-100.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following condition values for this routine:

```
ctcBadObjState
ctcInvParam
ctcMissingParam
ctcNotAllowed
ctcUnavailable
```

For details of the condition values that can be returned by the CTC/CMP API and CTC/CMP server, refer to Chapter 3.

ctcCmpSetAgentStatus

Set Status for an ACD Agent

Format in C

```
unsigned int ctcCmpSetAgentStatus
            (ctcChanId      channel,
             unsigned int   agentMode,
             ctcDeviceString agentData,
             ctcDeviceString password)
```

Description

The `ctcCmpSetAgentStatus` 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 (not ready to take calls)

Restrictions

This routine is not supported for channels assigned to telephones, ACD groups, CDNs, or monitor channels.

Arguments

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

This argument is a `ctcChanId` datatype that contains the channel ID value returned by `ctcCmpAssign` for the device in use.

ctcCmpSetAgentStatus

agentMode

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

This argument specifies the status for an ACD agent. It contains one of the following values:

Value	Description
ctcK_AgentReady	The agent is ready to receive calls
ctcK_AgentNotReady	The agent is not ready to receive inbound calls
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 agent data that you supply when an agent logs in or when they are no longer able to receive calls. Use this argument to specify:

- The agent's login identifier (ID) when the agent logs in. The login ID uniquely identifies the agent requesting association with a position ID.
- The reason why an agent has changed their work mode to ctcK_AgentNotReady. This optional.

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

If you do not need to specify agent data, pass the address of a zero-length character string with this argument.

password

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

This character string contains a password (if required) for logging in an agent.

The maximum length for password is specified by the literal ctcMaxDnLen in the CTC/CMP definitions file (see Section 1.6). Note that this maximum length includes the null termination character (NUL).

ctcCmpSetAgentStatus

Passing data with this argument is optional. If you do not need to specify a password, pass the address of a zero-length character string.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following routine-related condition values:

- `ctcAgentNotReady`
- `ctcAgentReady`
- `ctcBadObjState`
- `ctcInvAgentData`
- `ctcInvConfig`
- `ctcInvParam`
- `ctcInvPosition`
- `ctcLogoutPending`
- `ctcMissingParam`
- `ctcNotAllowed`
- `ctcNotLoggedIn`
- `ctcPasswordMismatch`
- `ctcSupOverride`
- `ctcUnavailable`

For details of the condition values that can be returned by the CTC/CMP API or CTC/CMP server for this routine, refer to Chapter 3.

ctcCmpSetMonitor

ctcCmpSetMonitor

Set Monitoring for a Device

Format in C

```
unsigned int ctcCmpSetMonitor
            (ctcChanId          channel,
             unsigned int      monitorMode)
```

Description

The `ctcCmpSetMonitor` routine changes the monitoring state of the assigned device.

You can use this routine with `ctcCmpGetEvent` to receive useful call data and agent status information associated with a device. This information is returned whenever a significant event occurs; for example, when an incoming call arrives, or when an active call is disconnected, or when an agent changes work mode.

Monitoring Devices

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

- The dialed number (the digits used to place the call)
- The extension numbers for those parties on the same switch
- For an outside call, if the telephone network can supply the information, the Automatic Number Identification (ANI) or Calling Line ID (CLID) that identifies the device that originated the call

Monitoring also returns a reference number for calls on the assigned device. This call reference identifies the call, and you specify it for most CTC/CMP routines used for processing calls.

Monitoring Groups

When you set monitoring ON for a primary or supplementary DN for an ACD group, CTC/CMP:

- Notifies you when a call enters or leaves the queue for the ACD group, and can tell you if the caller has disconnected or if the call has been routed to an agent.

ctcCmpSetMonitor

- Returns events for all agent positions associated with the group. Even if monitoring is not set ON for the agent positions, events for these positions are returned on the channel assigned to the group.

Monitoring an Agent Position and its Associated Group

If you are monitoring an agent position *and* its associated groups, event data is returned on the channel assigned to the agent position and on the channel assigned to the group whenever an event occurs for the agent position. However, the event data returned on each channel is different.

When you monitor a group, the CTC/CMP provides information about the switch's activities relating to the agent position, rather than actual activities at the agent position. For example, when a call is received by the agent, CTC/CMP returns:

- ctcK_InboundCall event on the channel assigned to the agent position
- ctcK_DestSeized event on the channel assigned to the group

The following table shows corresponding event and state values:

Returned Agent Position Values		Corresponding ACD Group Values	
Event	State	Event	State
ctcK_InboundCall	ctcK_ReceivedState	ctcK_DestSeized	ctcK_DeliverState
ctcK_TpAnswered	ctcK_ActiveState	ctcK_OpAnswered	ctcK_ActiveState
ctcK_TpDiscontinued	ctcK_NullState	ctcK_OpDisconnected	ctcK_NullState

To receive the same monitoring information for an agent position and its group, Dialogic recommend that you:

- Assign a channel to the agent position and use ctcCmpSetMonitor and ctcCmpGetEvent to receive event data.
- Assign to a monitor channel and use ctcCmpAddMonitor to monitor all of the agent positions in a group.

For more information, refer to the descriptions of ctcCmpAssign and ctcCmpAddMonitor in this chapter.

Restrictions

This routine is not supported for channels assigned to monitor channels.

ctcCmpSetMonitor

Arguments

channel

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

This argument is a `ctcChanId` datatype that contains the channel ID value returned by `ctcCmpAssign` 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
<code>ctcK_On</code>	Sets monitoring on for the device
<code>ctcK_Off</code>	Sets monitoring off for the device

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the CTC/CMP API or the CTC/CMP server.

The CTC/CMP server can return the following monitoring-related condition values for this routine:

`ctcMonAlreadyOn`
`ctcInvMonitorMode`
`ctcMonNotOn`

For details of the condition values that can be returned by the CTC/CMP API or CTC/CMP server for this routine, refer to Chapter 3.

ctcCmpSetRoutingEnable

Set the Routing State for a Device

Format in C

```
unsigned int ctcCmpSetRoutingEnable
            (ctcChanId  channel,
             unsigned int routingMode)
```

Description

The `ctcCmpSetRoutingEnable` routine enables or disables routing by the CTC/CMP server for the assigned CDN route point:

- If you enable routing, the switch passes a route request to CTC/CMP when it receives a call for the assigned CDN route point. You use `ctcCmpGetRouteQuery` to receive the route request and `ctcCmpRespondToRouteQuery` to provide a new route for the call.
- If you disable routing, by setting the routing mode to `ctcK_CmpDefault` or `ctcK_CmpRevertToDefault`, the switch stops sending route requests to CTC/CMP for the assigned CDN route point.

Enabling Call Routing

To enable call routing, you use the following sequence of routines:

1. `ctcCmpAssign` to assign a channel to the CDN route point.
2. `ctcCmpSetRoutingEnable` to explicitly enable call routing for the assigned CDN route point. The switch passes route requests to CTC whenever it receives a call at the CDN route point.
3. `ctcCmpGetRouteQuery` to receive the route requests.
4. `ctcCmpRespondToRouteQuery` to respond to the route requests.

For more information, refer to the descriptions of the `ctcCmpGetRouteQuery` and `ctcCmpRespondToRouteQuery` routines.

Disabling Call Routing

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

You can continue to receive information about calls made to the CDN route point by using `ctcCmpGetEvent`.

ctcCmpSetRoutingEnable

Restrictions

This routine is only supported for channels assigned to CDNs.

Arguments

channel

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

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

Specify the channel identifier returned by `ctcCmpAssign` for the CDN 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
<code>ctcK_CmpDefault</code>	Route incoming calls to the default ACD DN (as configured in the DMS-100). This sets the CDN to DEFAULT mode.
<code>ctcK_CmpControlled</code>	Route incoming calls to the CTC/CMP server. This sets the CDN to CONTROLLED mode.
<code>ctcK_CmpRevertToDefault</code>	Route incoming calls and existing calls in the CDN queues to the default ACD DN. This returns the CDN to DEFAULT mode.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

ctcCmpSetRoutingEnable

The DMS-100 can return the following condition values for this routine:

- ctcInvDN
- ctcInvParam
- ctcMissingParam
- ctcNotAllowed

For details of the condition values that can be returned by the CTC/CMP API and CTC/CMP server, refer to Chapter 3.

ctcCmpSnapshot

ctcCmpSnapshot

Query the Current State of a Device

Format in C

```
unsigned int ctcSnapshot (ctcChanId channel,  
                           unsigned int *state)
```

Description

The ctcCmpSnapshot routine returns information for the call at the assigned telephony device. ctcCmpSnapshot provides the state of each call.

Restrictions

This routine is supported for channels assigned to telephones only. For example, residential or Centrex telephones. For more information, refer to the description of ctcCmpAssign.

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 ctcCmpAssign for the device in use.

state

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

This argument contains the address of a 32-bit field that returns the state for a call.

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

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following condition values for this routine:

- `ctcInvDN`
- `ctcInvParam`
- `ctcNotAllowed`

For details of the condition values that can be returned by the CTC/CMP API and CTC/CMP server, refer to Chapter 3.

ctcCmpTransferCall

ctcCmpTransferCall

Transfer a Call

Format in C

```
unsigned int ctcCmpTransferCall
            (ctcChanId          channel,
             ctcCmpNetCallId   *callRefId)
```

Description

The `ctcCmpTransferCall` routine completes the transfer of a call (initiated by the `ctcCmpConsultationCall` routine) to a different device, and disconnects the assigned device.

For example, when agent A receives a call from B and wants to transfer that call to C:

1. A calls C, using `ctcCmpConsultationCall`, which automatically puts the call from B on hold.
2. A invokes `ctcCmpTransferCall` 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 `ctcCmpTransferCall`. For unscreened (or unsupervised) transfer, A invokes `ctcCmpTransferCall` before C answers the telephone.

Restrictions

This routine is not supported for channels assigned to ACD groups, CDNs, or monitor channels.

Arguments

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

This argument is a `ctcChanId` datatype that contains the channel ID value returned by `ctcCmpAssign` for the device in use.

ctcCmpTransferCall

callRefId

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

This argument contains the address of a fixed-format structure, for which you allocate memory of type `ctcCmpNetCallId`. The structure is defined in a CTC/CMP definitions file (see Section 1.6) and is formatted as follows:

```
struct ctcCmpNetCallId {  
    unsigned int    netNodeId;  
    unsigned int    localCallId;  
}
```

Information in the `ctcCmpNetCallId` structure provides a unique call reference identifier for the call. The DMS-100 returns the following information in the fields in the `ctcCmpNetCallId` structure:

- **netNodeId**

This 32-bit field returns a network node identifier. This is used to identify which DMS-100 first receives a call when a call is handled by more than one DMS-100.

- **localCallId**

This field returns a local call identifier. This is used to uniquely identify the call on the DMS-100.

Condition Values Returned

When the routine completes successfully, CTC/CMP returns the condition value `ctcSuccess`. If the routine does not complete successfully, CTC/CMP returns a condition value generated by the DMS-100, the CTC/CMP API, or the CTC/CMP server.

The DMS-100 can return the following routine-related condition values:

```
ctcBadObjState  
ctcInvParam  
ctcMissingParam  
ctcNotAllowed  
ctcNotLoggedIn  
ctcUnavailable
```

For details of the condition values that can be returned by the CTC/CMP API or CTC/CMP server for this routine, refer to Chapter 3.

ctcCmpTransferCall

Error and Condition Values Returned

Table 3–1 provides a brief description of the errors and condition values that can be returned by CTC/CMP routines.

Use Table 3–1 in conjunction with the `ctcCmpErrMsg` routine. This routine provides the name of the condition or error associated with a returned value, as defined in the files `ctcmperr.h` and `ctc_err.h`. For more information, refer to the description of the `ctcCmpErrMsg` routine in Chapter 2.

To help you determine and isolate problems, Table 3–1 shows the source of the reported condition: the CTC/CMP API, the CTC/CMP server, or the DMS-100.

The following general guidelines apply:

- Errors from the CTC/CMP API are usually returned for programming errors. For example, if you specify an invalid type or argument.
- Condition values from the CTC/CMP server are usually associated with resources, or CTC/CMP management.
- Condition values from the DMS-100 are often returned when there is a problem with the device state or the call reference. For example, when you provide an invalid call reference, or try to perform an operation and the device is in the wrong state for that operation.

Table 3–1 Condition Values Returned

Condition	From	Description
ctcAgentNotReady	DMS-100	The agent position is currently not in the ready state.
ctcAgentReady	DMS-100	The agent position is currently in the ready state.
ctcAsn1DecodeErr	CTC/CMP server	A bad message format record was received from the DMS-100.
ctcAsn1EncodeErr	CTC/CMP server	A bad parameter was supplied in the message to the ASN1 encoding routine.
ctcAssociated	DMS-100	The primary DN for the ACD group is already associated with another session.
ctcBadObjState	DMS-100	The object is in the incorrect state for the service. The switch is unable to provide more specific information.
ctcBindFail	CTC/CMP API	An RPC network binding handle cannot be created from the serverName and networkType arguments for ctcCmpAssign.
ctcCallAborted	DMS-100	The switch has indicated that the outbound call was aborted.
ctcComFail	CTC/CMP server	The communications initialization procedure detected insufficient virtual memory.
ctcCondError	CTC/CMP server	An internal error has occurred on the CTC/CMP server.
ctcCondWaiting	CTC/CMP server	An internal error has occurred on the CTC/CMP server.
ctcDeadLock	CTC/CMP server	An internal error has occurred on the CTC/CMP server.
ctcEventDataLost	CTC/CMP server	A number of events have occurred at the same time and some event data has been lost.
ctcEventInProgress	CTC/CMP server	The ctcCmpGetEvent routine has already been called.
ctcFileOpenError	CTC/CMP server	The CTC/CMP 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 <i>CT-Connect for CompuCALL Installation and Administration Guide</i> for details of the CTC/CMP Control Program TRACE command.
ctcInitFail	CTC/CMP server	The communications initialization procedure detected insufficient virtual memory.

Table 3–1 Condition Values Returned (Continued)

Condition	From	Description
ctcInsMem	CTC/CMP server	Insufficient virtual memory available, either on the CTC/CMP server or the CTC/CMP client, to complete the routine. Check your application's use of memory and ask your system manager to check the system parameters.
ctcInternError	CTC/CMP server	An unspecified internal error has occurred on the CTC/CMP server. Report the problem to Dialogic.
ctcInvAgentData	DMS-100	The agent login identifier is invalid.
ctcInvalidDest	DMS-100	The specified called party is invalid or unavailable.
ctcInvCallingDevice	DMS-100	The calling device is invalid.
ctcInvCallType	DMS-100	An invalid call type was specified with ctcCmpMakeCall.
ctcInvChannel	CTC/CMP API	An invalid channel identifier was specified. Specify the channel identifier as returned by the ctcCmpAssign routine. Note that, after a linkdown, the channel identifier on the CTC/CMP server becomes invalid. Under these circumstances, a call to any CTC/CMP routine - other than ctcCmpAssign - will return ctcInvChannel.
ctcInvCode	DMS-100	An invalid or unexpected account code or authorisation code was specified with ctcCmpMakeCall.
ctcInvConfig	DMS-100	The value specified with the AgentMode argument for the ctcCmpSetAgentStatus routine is invalid. For example, you specified ctcK_AgentNotReady, and the feature is not supported on the DMS-100.
ctcInvDN	DMS-100	The DN specified is invalid, unknown or is not configured.
ctcInvLogID	CTC/CMP API	The specified logical identifier is invalid or does not exist. Make sure you specify the same logical identifier as defined on the CTC/CMP server, which specifies the DMS-100 in use.
ctcInvMonitorMode	CTC/CMP API	The monitorMode argument for ctcCmpSetMonitor contains an invalid value.
ctcInvNetType	CTC/CMP API	The networkType argument for ctcCmpAssign contains an invalid or unsupported RPC protocol sequence string.

Table 3–1 Condition Values Returned (Continued)

Condition	From	Description
ctcInvParam	CTC/CMP API	One of the arguments specified with a CTC/CMP routine contains an invalid value.
ctcInvPosition	DMS-100	The agent state cannot be set with <code>ctcCmpSetAgentStatus</code> because the agent position is in a transient state. This could mean that the agent is currently trying to set their work mode using the keypad on their telephone.
ctcInvRouteData	CTC/CMP API	The route information specified is invalid.
ctcInvServerName	CTC/CMP API	The <code>serverName</code> parameter for <code>ctcCmpAssign</code> contains an invalid CTC/CMP server name or address string.
ctcLibFail	CTC/CMP server	The CTC/CMP server was unable to load the modules for the protocol specified in the CTC/CMP server startup by the Control Program SET LINK command. Report the problem to Dialogic.
ctcLinkConnectFail	CTC/CMP server	The connection for the link between the CTC/CMP server and the DMS-100 has failed.
ctcLinkDown	CTC/CMP server	The link between the CTC/CMP server and the DMS-100 is down.
ctcLogoutPending	DMS-100	The agent position is in a pending logged out state.
ctcMissingParam	DMS-100	One of the required argument is missing from the CTC/CMP routine.
ctcMonAlreadyOn	CTC/CMP API	Monitoring is already set on for this channel.
ctcMonitorOff	CTC/CMP API	Monitoring is set off for this channel so the call to <code>ctcCmpGetEvent</code> has been returned.
ctcMonNotOn	CTC/CMP API	Monitoring is not set on for this channel.
ctcMutexLocked	CTC/CMP server	An internal error has occurred on the CTC/CMP server.
ctcNoEvent	CTC/CMP server	The <code>dontWait</code> argument for <code>ctcCmpGetEvent</code> is set to TRUE and there is no event data at the CTC/CMP server for this channel.
ctcNoRoute	CTC/CMP server	There is no route data at the CTC/CMP server for this channel.
ctcNotAllowed	DMS-100	You are not allowed to use this function, for example, make outbound calls. The DN specified does not subscribe to the appropriate DMS-100 service functions.

Table 3–1 Condition Values Returned (Continued)

Condition	From	Description
ctcNotLoggedIn	DMS-100	The agent is not logged in to make an outbound call.
ctcNotOn	CTC/CMP server	Monitoring is not set on for this channel.
ctcNoUnlock	CTC/CMP server	An internal error has occurred on the CTC/CMP server.
ctcOrigTimeout	DMS-100	The agent failed to respond within the period defined on the DMS-100.
ctcOverallMonLimEx	DMS-100	Subscribed resource availability error. The request exceeded the switch's limit of monitors for the specified object.
ctcParseErr	CTC/CMP server	The CTC/CMP server could not parse the message from the DMS-100. This indicates an internal error. Report the problem to Dialogic.
ctcRcvReqRej	CTC/CMP server	The DMS-100 rejected a message or request from the CTC/CMP server. This indicates that the function is not supported on the switch.
ctcReadError	CTC/CMP server	The read data request on the link between the CTC/CMP server and the DMS-100 has returned an error. This could indicate that the DMS-100 has stopped or restarted the link, or that there may be a problem with the link hardware.
ctcResourceBusy	DMS-100	An internal resource is temporarily busy.
ctcRouteDataLost	CTC/CMP server	A number of calls have been presented to the application at the same time and some route data has been lost.
ctcRouteInProgress	CTC/CMP server	A previous ctcGetRouteQuery request has not yet completed.
ctcRoutingOff	CTC/CMP server	The call to ctcGetRouteQuery has been returned because the specified channel is deassigned.
ctcRPCConnecFail	CTC/CMP API	An RPC error was received. Due to the error, an RPC network connection cannot be created and the CTC/CMP client cannot communicate with the CTC/CMP server.
ctcServerUnknown	CTC/CMP API	The specified system is not a CTC/CMP server. Check that the specified name is correct and that the CTC/CMP server software is installed and running on the system.
ctcServerBusy	CTC/CMP API	The CTC/CMP server is too busy to respond, possibly because it is shutting down.

Table 3–1 Condition Values Returned (Continued)

Condition	From	Description
ctcSuccess	CTC/CMP API	The routine completed successfully.
ctcTimeout	CTC/CMP server	The DMS-100 did not respond to the request from the CTC/CMP server. There may be a problem with the link between the CTC/CMP server and the DMS-100 or the DMS-100 may be too busy to respond.
ctcUCBFail	CTC/CMP server	The UCB initialization procedure detected insufficient virtual memory on the CTC/CMP server.
ctcUnavailable	DMS-100	The resources required by this function are unavailable.
ctcUnsupAPIversion	CTC/CMP API	Either the version of CTC/CMP server software running on the CTC/CMP server is not compatible with the version of CTC/CMP API running on your CTC/CMP 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/CMP API	The specified procedure is not supported for the assigned device.
ctcXmitError	CTC/CMP server	The send data request on the link between the CTC/CMP server and the DMS-100 has returned an error condition.

Index

A

ACD group
 routines supported, 2-10

acdDn field
 ctcCmpEventData structure, 2-49
 ctcCmpRouteData structure, 2-64

acdGroup field
 ctcCmpRouteData structure, 2-64

Active state, 2-42

Add monitor
 ctcCmpAddMonitor, 2-2
 restrictions, 2-3

Agent position
 monitoring, 2-95
 routines supported, 2-10
 setting status, 2-91

Answering a call
 ctcCmpAnswerCall, 2-7

Application identifier, 2-47, 2-63

applicationData field
 ctcCmpEventData structure, 2-51
 ctcCmpRouteData structure, 2-66

Arguments
 optional data, 1-9
 order, 1-6
 passed by reference, 1-8
 passed by value, 1-8
 use, 1-6

Assigning a channel
 ctcCmpAssign, 2-9

C

Call
 events, 2-42
 routing, 2-59
 states, 2-41 to 2-42
 types, 2-46, 2-63

Call reference identifier
 monitoring, 2-94

calledParty field
 ctcCmpEventData structure, 2-48

calledPartyType field
 ctcCmpEventData structure, 2-48

callingParty field
 ctcCmpEventData structure, 2-48
 ctcCmpRouteData structure, 2-64

callingPartyName field
 ctcCmpEventData structure, 2-47

callingPartyType field
 ctcCmpEventData structure, 2-48
 ctcCmpRouteData structure, 2-63

callMode field
 ctcCmpEventData structure, 2-46

callType field
 ctcCmpEventData structure, 2-46
 ctcCmpRouteData structure, 2-63

Canceling a call
 ctcCmpCancelCall, 2-17

Channel
 assigning, 1-3
 deassigning, 2-25
 identifier, description, 2-12

- identifier, when to use, 2-9
- chargeNumber field
 - ctcCmpEventData structure, 2-49
 - ctcCmpRouteData structure, 2-64
- Compiling a program
 - Digital UNIX, 1-13
 - HP-UX, 1-14
 - OpenVMS, 1-14
 - OS/2, 1-15
 - SCO OpenServer, 1-15
 - Solaris, 1-15
 - Windows 95, 1-16
 - Windows NT, 1-16
- CompuCALL
 - term, ix
- Condition values, 3-1 to 3-6
 - ctcCmpErrMsg, 2-26
 - definitions file, 1-9
 - using, 1-10
- Conference calls
 - completing, 2-19
 - ctcCmpConferenceJoin, 2-19
 - ctcCmpConsultationCall, 2-21
 - initiating, 2-21
- Configuration Program, 2-14
- Constants, 1-9
 - definitions file, 1-9
- Consultation hold
 - ctcCmpConsultationCall, 2-21
 - disconnecting a call, 2-17
 - effects of ctcCmpHangupCall, 2-73
 - retrieving the call, 2-89
- CTC API
 - new features, 1-17
 - V2.0 features, 1-17
- ctcCmpAddMonitor, 2-2
 - restrictions, 2-3
- ctcCmpAnswerCall, 2-7
- ctcCmpAssign, 2-9
- ctcCmpCancelCall, 2-17
- ctcCmpConferenceJoin, 2-19
 - initiating, 2-21
- ctcCmpConsultationCall, 2-21
- ctcCmpDeassign, 2-25
 - and link down, 1-10
 - when to use, 2-25

- ctcCmpErrMsg, 2-26
- ctcCmpGetACDGroupStatus, 2-28
- ctcCmpGetAgentStatus, 2-33
- ctcCmpGetChannelInformation, 2-34
 - information returned by, 2-34
 - when to use, 2-34
- ctcCmpGetEvent, 2-38
 - creating a thread for, 1-12
 - lost event data, 2-39
- ctcCmpGetMessageWaiting, 2-55
- ctcCmpGetMonitor, 2-57
- ctcCmpGetRouteQuery, 2-59
- ctcCmpGetRoutingEnable, 2-69
- ctcCmpGiveTreatment, 2-70
- ctcCmpHangupCall, 2-73
- ctcCmpHoldCall, 2-75
- ctcCmpMakeCall, 2-77
 - and conferencing, 2-21
- ctcCmpReconnectHeld, 2-82
- ctcCmpRemoveMonitor, 2-84
- ctcCmpRespondToRouteQuery, 2-86
- ctcCmpRetrieveHeld, 2-89
- ctcCmpSetAgentStatus, 2-91
- ctcCmpSetMonitor, 1-4, 2-94
 - monitor channels, 2-10
- ctcCmpSetRoutingEnable, 2-97
- ctcCmpSnapshot, 2-100
- ctcCmpTransferCall, 2-102

D

Data structures

- definitions file, 1-9
- description, 1-6

Data types, 1-6

- ctcChanData, 1-7
- ctcChanId, 1-7, 2-12
- ctcCmpApplString, 1-7
- ctcCmpAssignData, 1-7
- ctcCmpCodeString, 1-7
- ctcCmpEventData, 1-7
- ctcCmpGetACDDData, 1-7
- ctcCmpNetCallId, 1-7
- ctcCmpRouteData, 1-7
- ctcDeviceString, 1-7
- ctcLogIdString, 1-8

- ctcNameString, 1-8
- ctcNetString, 1-8
- ctcTimeStamp, 1-8
- structures, 1-6

DCE Thread Library, 1-13

Deassigning a channel

- ctcCmpDeassign, 2-25

Definitions files

- condition values, 1-9
- constants, 1-9
- data structures, 1-9
- location, 1-9

Deliver state, 2-42

Device

- assigning channels to, 2-12
- DN, 2-34
- identifying, 2-9, 2-12

dialedParty field

- ctcCmpEventData structure, 2-49
- ctcCmpRouteData structure, 2-64

Digital UNIX

- compiling and linking programs, 1-13

Disconnecting a call

- ctcCmpCancelCall, 2-17
- ctcCmpHangupCall, 2-73

E

End of call state, 2-42

Error messages

- See* Condition values

event field

- ctcCmpEventData structure, 2-42

eventQualifier field

- ctcCmpEventData structure, 2-44

Events, 2-42 to 2-44

- agent events, 2-43, 2-44
- call events, 2-43
- data lost, 2-2, 2-39
- qualifiers, 2-44

Exception handling, 1-11

F

Fail state, 2-42

Feature phone, 2-77

- hands-free answering, 2-7

G

Get routines

- ctcCmpGetACDGroupStatus, 2-28
- ctcCmpGetChannelInformation, 2-34
- ctcCmpGetEvent, 2-38
- ctcCmpGetMessageWaiting, 2-55
- ctcCmpGetMonitor, 2-57
- ctcCmpGetRouteQuery, 2-59

Groups

- monitoring, 2-94

H

Hanging up a call

- ctcCmpHangupCall, 2-73

Held calls

- retrieving, 2-89

Hold

- and ctcCmpHangupCall, 2-73
- ctcCmpHoldCall, 2-75
- putting a call on, 2-75
- state, 2-42

HP-UX

- compiling and linking programs, 1-14

I

Idle state

- See* Null state

Incoming calls

- state changes, 2-41

Initiate state, 2-42

L

Line type, 2-34

Link

- gone down, 1-10
- logical identifier, 2-14

Linking a program

- Digital UNIX, 1-13
- HP-UX, 1-14
- OpenVMS, 1-14
- OS/2, 1-15
- SCO OpenServer, 1-15
- Solaris, 1-15
- Windows 95, 1-16
- Windows NT, 1-16

Literal, 1-10

localCallId field

- ctcCmpNetCallId structure, 2-8, 2-18, 2-20, 2-23, 2-41, 2-62, 2-71, 2-74, 2-76, 2-80, 2-83, 2-90, 2-103

Logical agents, 2-92

Logical identifier

- specifying, 2-14

Lost event data, 2-39

M

Making calls

- ctcCmpMakeCall, 2-77

Masks, 1-10

Meridian SCAI, vii

Monitor channels, 1-2

- and ctcCmpSetMonitor, 2-10
- ctcCmpAddMonitor, 2-2
- ctcCmpRemoveMonitor, 2-84
- monitoring other monitor channels, 2-3, 2-13
- routines supported, 2-10

Monitoring, 2-94

- ACD groups, 2-38, 2-94
- ACD queues, 2-94
- call queues, 2-94
- devices, 2-38, 2-94

- events, 2-42

- for incoming call, 2-7

- groups, 2-38, 2-94

- information, 2-57

- logical entities, 2-38

- monitor channels, 2-3, 2-13, 2-38

- off, 2-96

- on, 2-96

- queues, 2-38

monitorParty field

- ctcCmpEventData structure, 2-52

Multithreaded programs

- creating, 1-12

- description, 1-11

- when to use, 1-11

- with CTC/CMP, 1-12

N

nestedMonitorChannel field

- ctcCmpEventData structure, 2-52

netNodeId field

- ctcCmpNetCallId structure, 2-8, 2-18, 2-20, 2-23, 2-41, 2-62, 2-71, 2-74, 2-76, 2-80, 2-83, 2-90, 2-103

Network problems

- exception-handling, 1-11

Nortel

- CompuCALL interface, vii

- DMS-100, vii

- Meridian SCAI interface, vii

- SL-100, vii

Null state, 2-42

O

Off-hook, 2-42

On-hook dialing

- full, 2-77

OpenVMS

- compiling and linking programs, 1-14

Options file, 1-14

OS/2

- compiling and linking programs, 1-15

Outgoing calls
state changes, 2-41

P

Passing mechanism
and optional arguments, 1-9
by reference, 1-8
by value, 1-8
prevAppId field
ctcCmpEventData structure, 2-47
ctcCmpRouteData structure, 2-63
Programs
linking, 1-13 to 1-16
multithreaded, 1-16

Q

Queued state, 2-42
Queues
monitoring, 2-94

R

Receive state, 2-42
Reconnecting a held call
ctcCmpReconnectHeld, 2-82
refId field
ctcCmpEventData structure, 2-41
ctcCmpRouteData structure, 2-62
Remove monitor
ctcCmpRemoveMonitor, 2-84
Responding to route queries
ctcCmpRespondToRouteQuery, 2-86
Retrieving a call on hold
ctcCmpRetrieveHeld, 2-89
Route determined
See Deliver state
Route points, 2-59
Routines, 2-1 to 2-103
access to data, 1-8
arguments, 1-6
call sequence, 1-5
ctcCmpAddMonitor, 2-2
ctcCmpAnswerCall, 2-7

ctcCmpAssign, 2-9
ctcCmpCancelCall, 2-17
ctcCmpConferenceJoin, 2-19
ctcCmpConsultationCall, 2-21
ctcCmpDeassign, 2-25
ctcCmpErrMsg, 2-26
ctcCmpGetACDGroupStatus, 2-28
ctcCmpGetAgentStatus, 2-33
ctcCmpGetChannelInformation, 2-34
ctcCmpGetEvent, 2-38
ctcCmpGetMessageWaiting, 2-55
ctcCmpGetMonitor, 2-57
ctcCmpGetRouteQuery, 2-59
ctcCmpGetRoutingEnable, 2-69
ctcCmpGiveTreatment, 2-70
ctcCmpHangupCall, 2-73
ctcCmpHoldCall, 2-75
ctcCmpMakeCall, 2-77
ctcCmpReconnectHeld, 2-82
ctcCmpRemoveMonitor, 2-84
ctcCmpRespondToRouteQuery, 2-86
ctcCmpRetrieveHeld, 2-89
ctcCmpSetAgentStatus, 2-91
ctcCmpSetMonitor, 2-94
ctcCmpSetRoutingEnable, 2-97
ctcCmpSnapshot, 2-100
ctcCmpTransferCall, 2-102
format, 1-6
how to call, 1-11
in a multithreaded program, 1-12
overview, 1-3 to 1-15
passing mechanism, 1-8
status returns, 1-10
synchronous operation, 1-11
Routing
ctcCmpGetRouteQuery, 2-59
ctcCmpRespondToRouteQuery, 2-86
ctcCmpSetRoutingEnable, 2-97
description, 2-59
new route data, 2-67

S

SCO OpenServer
compiling and linking programs, 1-15

Screened transfer, 2-102
Set routines
 ctcCmpSetAgentStatus, 2-91
 ctcCmpSetMonitor, 2-94
 ctcCmpSetRoutingEnable, 2-97
SL-100, vii
Snapshot the state of a device
 ctcCmpSnapshot, 2-100
Software
 versions, vii
Solaris
 compiling and linking programs, 1-15
state field
 ctcCmpEventData structure, 2-41
States
 changing, 2-41
 described, 2-41
 monitoring, 2-94
Status returns, 1-10
Structure
 description, 1-6

T

Telephone set
 routines supported, 2-10
Telephony functions, 1-4
Thread stack size
 Windows 95 programs, 1-16

Windows NT programs, 1-16
Threads
 and data passing, 1-12
 and route data, 2-67
 description, 1-11
 execution, 1-12
timeStamp field
 ctcEventData structure, 2-53, 2-67
Transferring a call
 ctcCmpTransferCall, 2-102
 initiating, 2-21
 screened and unscreened, 2-102

U

Unavailable state, 2-42
Unscreened transfer, 2-102

V

Versions of software, vii

W

Windows 95
 compiling and linking programs, 1-16
Windows NT
 compiling and linking programs, 1-16