

Intel® NetMerge™
Call Processing Software

CTC Test User's Guide

Software/Version: Intel NetMerge Call Processing Software Version 6.0

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

This document as well as the software described in it is furnished under license and may only be used or copied in accordance with the terms of the license. The information in this manual is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Intel Corporation. Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document.

Except as permitted by such license, no part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without express written consent of Intel Corporation.

Copyright © 2002 Intel Corporation.

Intel and NetMerge are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Acrobat and Adobe are trademarks of Adobe Systems, Incorporated.

†Other names and brands may be claimed as the property of others.

Publication date: September, 2002

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

Contents

| | |
|--------------------------------|---|
| About This Manual | v |
|--------------------------------|---|

1 Overview

| | |
|---|-----|
| 1.1 Purpose of the CTC Test Program | 1-1 |
| 1.2 Prerequisites | 1-1 |
| 1.3 Where Can You Use CTC Test? | 1-2 |
| 1.4 Types of Testing | 1-2 |
| 1.5 Preparing and Running a Test | 1-3 |
| 1.6 Getting Help While Running CTC Test | 1-3 |

2 How to Start Your Test

| | |
|--|-----|
| 2.1 Starting CTC Test | 2-1 |
| 2.2 Running Your Tests | 2-2 |
| 2.2.1 Running Individual CTC Test Commands | 2-2 |
| 2.2.2 Running a CTC Test Script | 2-2 |

3 Test Functions and Commands

| | |
|---|-----|
| 3.1 Scope and Conventions | 3-1 |
| 3.1.1 How to Allocate a Channel | 3-1 |
| 3.1.2 Command Sequences | 3-2 |
| 3.1.3 Syntax Conventions | 3-3 |
| 3.2 Matching Functions to Commands | 3-3 |
| 3.3 Command Descriptions and Syntax | 3-8 |

4 Writing a CTC Test Script

| | |
|-------------------------------------|-----|
| 4.1 Test Script Overview | 4-1 |
| 4.2 Script Control Qualifiers | 4-2 |
| 4.2.1 Description | 4-2 |
| 4.2.2 Functions Available | 4-3 |
| 4.2.3 Definitions | 4-3 |
| 4.3 Script Structure Commands | 4-6 |
| 4.3.1 Description | 4-6 |
| 4.3.2 Functions Available | 4-6 |

| | | |
|-------|----------------------------------|------|
| 4.3.3 | Definitions | 4-7 |
| 4.4 | Macro Substitution | 4-10 |
| 4.4.1 | Declare | 4-11 |
| 4.4.2 | Define | 4-11 |
| 4.4.3 | Invoke | 4-11 |
| 4.4.4 | Example Use of a Macro | 4-12 |

A Example CTC Test Scripts

| | | |
|-----|---------------------------------------|-----|
| A.1 | Example CTC Test Script | A-1 |
| A.2 | Example CTC Test Macro File | A-2 |

About This Manual

This manual describes how to use the Call Processing Software CTC Test program.

Audience

This manual is intended for anyone who wishes to use CTC Test to check for problems on an operational Call Processing Software system or to test a call processing application. It assumes that readers are familiar with Call Processing Software concepts and the call processing Application Programming Interface (API) functions.

Associated Documentation

Intel NetMerge Call Processing Software Documentation

In addition to this manual, the following are included in the Intel NetMerge Call Processing Software documentation set:

- *Intel NetMerge Call Processing Software Introduction*—This manual provides an overview of the Call Processing Software and example network configurations. It is provided in hard copy and as a Portable Document Format (PDF) file.
- *Intel NetMerge Call Processing Software Installation and Configuration Guide*—This manual describes how to install and configure the call processing server and the call processing API software on supported platforms, and how to start and stop the call processing server. This manual is provided in hard copy and as a Portable Document Format (PDF) file.
- *Intel NetMerge Call Processing Software C Programming Guide*—This manual provides detailed descriptions of the API procedural routines and guidelines for using them. It also includes details of the operational differences for links to specific switches. This manual is provided in hard copy and as a PDF file.
- *Intel NetMerge Call Processing API for the Java™ Platform*—This information provides a detailed description of the API for the Java™ platform. This defines the Call Processing Software packages, interfaces, and classes, along with the fields and methods supported. It also includes details of the operational differences for links to specific switches. This information is provided as a series of HTML files.

- *Intel NetMerge Call Processing Software Management API C Programming Guide*—This manual describes all management API routines, and provides guidelines for creating a management application similar to the Control Program. This manual is provided only as a PDF file.
- *Intel NetMerge Call Processing Software Management API for the Java™ Platform*—This information provides a detailed description of the management API for the Java platform. This defines the Call Processing Software packages, interfaces, and classes, along with the fields and methods supported. This information is provided as a series of HTML files.
- *Intel NetMerge Call Processing Software Release Notes*—These online notes provide information about changes to the software and/or documentation at the time of release. This document is provided as a text file.

All online files, and the Adobe™ Acrobat™ Reader used to view PDF files, can be installed at the same time as the call processing server software. For details, refer to the *Intel NetMerge Call Processing Software Installation and Configuration Guide*.

Switch Documentation

Refer to the documentation supplied with the switch for details of its features and characteristics.

Terms and Definitions

The following terms are used throughout this manual:

| Term | Definition |
|---|---|
| Call processing API for the Java platform | The Intel NetMerge Call Processing Software API for the Java platform. |
| Call processing API | The Intel NetMerge Call Processing Software API for all supported platforms |
| Client or call processing client | A supported system that has the call processing API software installed, and is running a call processing application. |
| Server or call processing server | A Windows NT†, Windows‡ 2000 or Windows XP personal computer running the call processing server software. |
| Communications link | A logical link between the server and the switch. |

| Term | Definition |
|----------------------------|--|
| CTC Test | The CTC Test program. |
| Switch | The telephony switching device. For example, a Private Branch Exchange (PBX), Private Automatic Branch Exchange (PABX), or central office switch. |
| Tru64 UNIX | Refers to the Compaq† Tru64† UNIX† operating system. |
| OpenVMS | Refers to the OpenVMS† Alpha operating system. |
| Windows 9x | Refers to the Windows 95, Windows 98 and Windows Me operating systems. |
| Programs on the Start Menu | Refers either to the Programs item on the Start menu on Window 9x, Windows NT and Windows 2000 systems, or to the All Programs item on the Start menu on Windows XP systems. |

Conventions

The following conventions are used throughout 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. |

Overview

1.1 Purpose of the CTC Test Program

CTC Test is a Call Processing Software program that you run on a client system. It can perform the sequence of actions you expect your application to take against a supported switch.

CTC Test provides you with functions to help:

- Isolate problems that occur during development of an application using the call processing Application Programming Interface (API).
- Test the validity of application call sequences within a specific Call Processing Software configuration.
- Verify functions supported by the switch.
- Verify events supported by the switch.

CTC Test allows you to perform these types of action:

- Assign a logical channel between a process on your call processing client, the call processing server, and a device on the switch.
- Perform common telephony actions, such as making a call.
- Display event information on calls that involve the assigned device.

1.2 Prerequisites

The information provided here assumes that you are already familiar with the Call Processing Software product and have experience of managing a Call Processing Software network.

Before you begin testing, ensure that your network is configured as it is intended to be when in operation.

1.3 Where Can You Use CTC Test?

You can use CTC Test on these platforms:

HP-UX†
OpenVMS
Solaris†
Tru64 UNIX
Windows 95
Windows 98
Windows NT
Windows 2000
Windows XP

You can use CTC Test in this environment:

Java 2 Runtime Environment (JRE)

1.4 Types of Testing

Use CTC Test to perform two types of testing:

- Execution of single CTC Test commands
You can enter single CTC Test commands to check quickly the correct functioning of your call processing application. For example, you might do this to track down the source of a problem on a running system.
- Execution of a test script containing multiple CTC Test commands
You can write a test script that executes combinations of CTC Test commands to rigorously test your call processing application. (Note that you cannot use test scripts if you are running CTC Test in the Java Runtime Environment.) For example, you might do this to perform regression, soak, or stress testing during application development and debugging.

1.5 Preparing and Running a Test

Follow these steps to prepare and run your tests:

1. Gather the information required by the CTC Test commands you intend to use (for details, see Chapter 3).
2. If you intend running a test script, prepare the script file (for details, see Chapter 4. For a sample script, see Appendix A).
3. Start CTC Test and run your test (for details, see Chapter 2).

1.6 Getting Help While Running CTC Test

To display online help when running CTC Test, do either of the following:

- Enter this command at the prompt:

```
help
```

This displays the Table of Contents for CTC Test help. Double-click the book icon to see a complete list of CTC Test commands.

- Enter this command at the prompt:

```
help command
```

where *command* is the command on which you want help. This displays help for the command.

How to Start Your Test

2.1 Starting CTC Test

To start CTC Test, use the command or procedure indicated for your operating system platform or environment (changing directory to where you have installed the CTC Test files, if necessary).

Operating System Platforms

| Platform | Command |
|---|--|
| HP-UX | <code>ctc_test</code> |
| OpenVMS | <code>RUN SYS\$TEST:CTC_TEST</code> |
| Solaris | <code>ctc_test</code> |
| Tru64 UNIX | <code>ctc_test</code> |
| Windows 9x, Windows NT and Windows 2000 | From the Start Menu, select Programs → Intel NetMerge Call Processing API → CTC Test |
| Windows XP | From the Start Menu, select All Programs → Intel NetMerge Call Processing API → CTC Test |

When CTC Test is running, the prompt looks like this:

```
ctcTest>
```

Environments

| Environment | Command or Procedure |
|------------------------------|------------------------------------|
| JRE from the Java 2 Platform | <code>java -jar ctctest.jar</code> |

When CTC Test is running, the prompt looks like this:

```
ctcJavaTest>
```

Note: Ignore any references to CTC Test scripts in this chapter if you are running in the Java Runtime Environment. CTC Test does not support the use of scripts in the Java Runtime Environment.

2.2 Running Your Tests

2.2.1 Running Individual CTC Test Commands

You run an individual CTC Test command by entering it at the `ctcTest>` or `ctcJavaTest>` prompt.

2.2.2 Running a CTC Test Script

You run a CTC Test script by entering this command at the `ctcTest>` prompt:

```
ctcTest> script script_file_name [qualifier]
```

where *script_file_name* is the name of your CTC Test script file.

Optionally, add qualifiers to further control the behavior of the script, as follows:

| | |
|---------------------------------------|--|
| <code>/iterations=<i>n</i></code> | Execute the entire script a set number of times. Specify the number of iterations in <i>n</i> . For infinite iterations, specify <i>n</i> as 0. Caution: Take care specifying <i>n</i> as 0 as this can rapidly consume system resources, particularly combined with other qualifiers. For example, setting <code>/loglevel</code> to 4 and recording the log in a file will quickly occupy large amounts of disk space. This overrides the <code>iterations</code> command within the script. |
| <code>/logfile=<i>filename</i></code> | Record log information in a file. Specify the destination file in <i>filename</i> . Note that CTC Test continues to display log information on screen. |

`/loglevel=n` Set the level of log information CTC Test outputs.
Specify one of the following values in *n*:

| | |
|---|--|
| 1 | Only report errors and test or iteration completion |
| 2 | As for level 1, plus minimal ctcGetEvent information |
| 3 | As for level 2, plus extended ctcGetEvent information |
| 4 | As for level 3, plus a trace of the commands (default) |

`/macros=filename` Tell CTC Test which macros file to reference for macro substitution.
Specify the macros file in *filename*.

`/parse` Stop script execution after parsing to obtain debug information without running tests.
During the parse, CTC Test checks for:

- Properly bounded routines (that is, matched pairs of braces { })
- Labels for all cleanup routines
- Valid verbs, qualifiers, and event names in CTC Test commands
- Definitions for all declared macros in the referenced macros file

`/stop` Stop execution of the script on receipt of any error.
This overrides any flags set within the script.

`/timeout=n` Set the default event timeout for the script.
Specify the number of seconds in *n*.
If you do not use this qualifier, CTC Test uses an event timeout of 10 seconds.

Test Functions and Commands

3.1 Scope and Conventions

The information provided here gives only a summary guide to the CTC Test commands and does not describe the numerous switch-specific restrictions that can apply. For detailed information, refer either to the *Intel NetMerge Call Processing Software C Programming Guide* or to the *Intel NetMerge Call Processing Software for the Java Platform* online documentation.

In this manual, the information on the CTC Test commands is listed in two ways:

1. By the test function performed, matched to the relevant command (see Section 3.2).
2. In command alphabetical order, including command descriptions and syntax (see Section 3.3).

Before reading this information, and before using the CTC Test commands, please familiarize yourself with the following:

- How to allocate a channel (see Section 3.1.1).
- The expected sequence of test commands (see Section 3.1.2).
- The syntax conventions used in the test command definitions (see Section 3.1.3).

3.1.1 How to Allocate a Channel

You must first allocate a channel between your call processing client and the target device.

- With the C API, use the `assign` or the `secassign` command to allocate the channel. If you use the `secassign` command, you will need to enter a username and password before the command can be executed.
- With the Java API, use the `create` or `assign` command to allocate the channel. You cannot use the `secassign` command.

For simplicity, this chapter uses only the `assign` command in its examples. Please

refer to the description of the `create` command and the `secassign` command for details of how they differ from the `assign` command.

CTC Test can handle up to 20 channels simultaneously. The `assign` command allows you to label channels (and therefore target devices) with a number to distinguish between them when issuing test commands. For example, in the following command, the channel is labelled 2:

```
ctcTest> 2 assign dn:333 system1 ctc_csta ncacn_ip_tcp
```

Note that the label is not the same as the channel identifier that is returned by some CTC Test commands (such as `channelinformation`).

If you do not specify a label, CTC Test uses the default of 1. For example, CTC Test performs an `answercall` to the designated device on the channel labelled 1 in both of the following:

```
ctcTest> 1 answercall
```

```
ctcTest> answercall
```

Hence, the general format of a CTC Test command is as follows:

```
ctcTest> [label] command parameters
```

In the syntax definitions, the label is not shown.

3.1.2 Command Sequences

Typically, you enter CTC Test commands in a sequence with other CTC Test commands. For example, to set up a conference call, you would enter commands as follows:

1. A `makecall` command to call the first party.
2. When the called party answers, a `consultationcall` command to call the second party.
3. When the second party answers, a `conferencejoin` command to bring all parties into the conference.

Refer to the programming guides for details of required command sequences.

3.1.3 Syntax Conventions

Note the conventions used in the syntax definitions:

- **deassign** characters indicate the minimum you must enter for CTC Test to recognize the command. This allows you to shorten commands. For example, the following shows that you need only enter `dea` for CTC Test to execute the command:

```
deassign
```

- Braces { } surround a list of parameters, with a bar | between each, one of which you must choose. For example, the following shows that you must choose one of the parameters {dn|route|monitor|id|voice|agent}:

```
removemonitor {dn|route|monitor|id|voice|agent}
```

- Square brackets [] indicate optional parameters you can specify. For example, the following shows that it is optional whether you add /data:

```
getagentstatus [/data=...]
```

- *Courier* is used to show characters as they will actually appear on screen. *Courier italic* is used for variables where you provide the values. For example, the following shows you enter the command `hammer` and specify a value for `dn` and `iterations`:

```
hammer dn iterations
```

Follow these general rules when entering text:

- Specify all commands and parameters in lowercase.
- Specify variables in either lowercase or uppercase.
- Some qualifiers are case sensitive. Where this is the case, it is highlighted in the command description. Enter them precisely as noted in the command description.
- Enter data in a command as a single, unbroken string. Do not use quotation marks to mark the start and end of the data or include spaces within the data.

3.2 Matching Functions to Commands

Look in the first column for the test function you want to perform and in the second column to find the name of the CTC Test command to use. Note that some CTC Test commands are specific to the Java environment. Do not use these commands in the C environment because CTC Test will fail. Similarly, do not use commands marked C only in the Java environment.

| To perform this function... | Use this command... |
|--|------------------------------------|
| Add an agent device to a monitor channel for monitoring (Java only) | <code>addagentdevicemonitor</code> |
| Add a DN device to a monitor channel for monitoring (Java only) | <code>adddevicemonitor</code> |
| Set monitoring on and obtain information on activity on the assigned device (Java only) | <code>addeventlistener</code> |
| Add a device to a monitor channel for monitoring | <code>addmonitor</code> |
| Add a physical identifier to a monitor channel for monitoring (Java only) | <code>addphysicalidmonitor</code> |
| Set monitoring on and obtain information on activity on the assigned route point (Java only) | <code>addroutelister</code> |
| Add a route point to a monitor channel for monitoring (Java only) | <code>addroutemonitor</code> |
| Answer a call on a hands-free set | <code>answercall</code> |
| Disconnect the specified party from a call | <code>asaidropparty</code> |
| Return ACD split information | <code>asaigetacdstatus</code> |
| Return the global reference identifier for a call | <code>asaigetglobalrefid</code> |
| Temporarily disconnect a party from a call so that the party can no longer hear one or more of the other parties on the call | <code>asaiselectivelisten</code> |
| Assign a communications channel | <code>assign</code> |
| Associate data with a call and pass it to the switch | <code>associatedata</code> |
| Associate data with a device at a remote switch connected to a remote call processing server | <code>associateremotedata</code> |
| Disconnect a consultation call | <code>cancelcall</code> |
| Bring participants into a conference call | <code>conferencejoin</code> |
| Place a consultation call to another device | <code>consultationcall</code> |
| Create a communications channel (Java only) | <code>create</code> |
| Deassign a communications channel | <code>deassign</code> |
| Deflect a call ringing on one device to another device | <code>deflectcall</code> |
| Enable exchange of private data with the switch | <code>escape</code> |
| Exit the CTC Test program | <code>exit</code> |
| Obtain the operating mode of the ACD agent | <code>getagentstatus</code> |
| Get information about one or more agents | <code>getallagentsstates</code> |

| To perform this function... | Use this command... |
|---|---------------------------------------|
| Obtain all of the call forward settings for the assigned device | <code>getallcallforwards</code> |
| Get information about the apparatus at the assigned device | <code>getauditoryapparatusinfo</code> |
| Get the Auto Answer setting at the assigned device | <code>getautoanswer</code> |
| Get information about the specified button at the assigned device | <code>getbuttoninfo</code> |
| Obtain the call forward setting for incoming calls on the assigned device | <code>getcallforward</code> |
| Get data on a communications channel and the device to which it is assigned | <code>getchannelinformation</code> |
| Get information about the display at the assigned device | <code>getdisplayinfo</code> |
| Obtain the do-not-disturb setting on the assigned device | <code>getdonotdisturb</code> |
| Obtain information on activity on the assigned device (C only) | <code>getevent</code> |
| Get information about the hookswitch status (on or off) at the assigned device | <code>gethookswitchstatus</code> |
| Get information about the specified lamp on the assigned device | <code>getlampinfo</code> |
| Get characteristics of a lamp at the assigned device, for example, its color and brightness | <code>getlampmode</code> |
| Obtain the message waiting indicator setting on the assigned device | <code>getmessagewaiting</code> |
| Get the gain setting for the microphone at the assigned device | <code>getmicrophonegain</code> |
| Get the mute setting for the microphone at the assigned device | <code>getmicrophonemute</code> |
| Obtain the monitoring state for the assigned device | <code>getmonitor</code> |
| Retrieve private data | <code>getprivatedata</code> |
| Retrieve private data associated with the last event returned on a channel | <code>getprivateeventdata</code> |
| Retrieve private data associated with the last route request received on a channel | <code>getprivateroutedata</code> |
| Get information about the ringer at the assigned device | <code>getringerstatus</code> |
| Return a route request for a call when it reaches a route point (C only) | <code>getroutequery</code> |
| Obtain routing enabled status | <code>getroutingenable</code> |
| Obtain information about the speaker mute setting at the assigned device | <code>getspeakermute</code> |

| To perform this function... | Use this command... |
|--|---------------------------------------|
| Obtain information about the speaker volume setting at the assigned device | <code>getspeakervolume</code> |
| Loop 'make call' and 'hang up' to a telephone | <code>hammer</code> |
| Clear a call on a device and return it to the null state | <code>hangupcall</code> |
| Display help text for a command | <code>help</code> |
| Put a call on hold | <code>holdcall</code> |
| Make a call | <code>makecall</code> |
| Allow a virtual party on a switch to initiate calls | <code>makepredictivecall</code> |
| Close an open voice file on a Meridian† Mail system | <code>mlpclosevoicefile</code> |
| Collect DTMF digits entered by a user | <code>mlpcollectedigits</code> |
| Log off from a Meridian Mail account | <code>mlplogoffmailbox</code> |
| Log on to a Meridian Mail account | <code>mlplogonmailbox</code> |
| Open a voice file on a Meridian Mail system | <code>mlpopenvoicefile</code> |
| Play a message to a caller | <code>mlpplaymessage</code> |
| Move a call from a telephony device to a parking lot defined on the switch | <code>parkcall</code> |
| Answer a call which is ringing at another device | <code>pickupcall</code> |
| Emulate pressing a button | <code>pressbutton</code> |
| Quit the CTC Test program (C only) | <code>quit</code> |
| Disconnect a consultation call and reconnect a held call | <code>reconnectcall</code> |
| Remove an agent position from a monitor channel (Java only) | <code>removeagentdevicemonitor</code> |
| Remove a device from a monitor channel (Java only) | <code>removedevicemonitor</code> |
| Set monitoring off (Java only) | <code>removeeventlistener</code> |
| Remove a device from a monitor channel | <code>removemonitor</code> |
| Remove a physical identifier from a monitor channel | <code>removephysicalidmonitor</code> |
| Remove a route point from a monitor channel (Java only) | <code>removeroutemonitor</code> |
| Set response if there is an existing call on the destination device | <code>respondtoinactivecall</code> |
| Provide a destination for a call in response to receipt of route request | <code>respondtoroutequery</code> |
| Retrieve a call that is on hold | <code>retrievecall</code> |

| To perform this function... | Use this command... |
|---|-----------------------------------|
| Provide a destination for a call in response to receipt of route request (Java only) | <code>routecall</code> |
| Assign a communications channel with a password as security protection (C only) | <code>secassign</code> |
| Transmit DTMF tones | <code>senddtmf</code> |
| Set status for an ACD agent | <code>setagentstatus</code> |
| Set the Auto Answer feature on or off | <code>setautoanswer</code> |
| Set information about the specified button | <code>setbuttoninfo</code> |
| Redirect incoming calls to another device | <code>setcallforward</code> |
| Set monitoring on or off for channels assigned to device type <code>asaicall</code> . | <code>setcallmonitor</code> |
| Set the contents of a physical display | <code>setDisplayinfo</code> |
| Set do-not-disturb on or off | <code>setdonotdisturb</code> |
| Set the hookswitch status on or off | <code>sethookswitchstatus</code> |
| Set the mode and features of a lamp at the assigned device | <code>setlampmode</code> |
| Set message waiting indicator on or off | <code>setmessagewaiting</code> |
| Set the gain (input level) for a microphone associated with the auditory apparatus | <code>setmicrophonegain</code> |
| Set the microphone mute on or off on the associated apparatus | <code>setmicrophonemute</code> |
| Set monitoring on or off | <code>setmonitor</code> |
| Send private data | <code>setprivatedata</code> |
| Set options for the specified ringer | <code>setringerstatus</code> |
| Set routing for an assigned route point on or off | <code>setroutingenable</code> |
| Set the speaker mute (on or off) on the associated apparatus | <code>setspeakermute</code> |
| Set the volume for a speaker at the auditory apparatus | <code>setspeakervolume</code> |
| Immediately join a call to a conference | <code>singlestepconference</code> |
| Transfer current call to another number and disconnect the assigned device | <code>singlesteptransfer</code> |
| Get the status and number of calls at the assigned device | <code>snapshot</code> |
| Swap the current call with a call on consultation hold | <code>swapwithheld</code> |

| To perform this function... | Use this command... |
|--|---------------------------|
| Complete the transfer of a call initiated by a <code>consultationcall</code> command | <code>transfercall</code> |

3.3 Command Descriptions and Syntax

addagentdevicemonitor

Environment: Java only

Syntax: `addagentdevicemonitor dn`

Adds an agent position to a monitor channel for monitoring (only for Nortel Meridian switches). Specify the agent position as *dn*.

adddevicemonitor

Environment: Java only

Syntax: `adddevicemonitor dn`

Adds a DN device to a monitor channel for monitoring.

addeventlistener

Environment: Java only

Syntax: `addeventlistener`

Sets monitoring on and obtains information on activity on the assigned device.

addmonitor

Environment: C and Java

Syntax: `addmonitor [{dn|route|monitor|id|voice|agent|asai} :] dn`

Adds a device to a monitor channel for monitoring.

Enter one of the following to indicate the type of device you want to monitor:

| | |
|----------------------|--|
| <code>dn</code> | A telephone or other telephony device (default if no device type is specified) |
| <code>route</code> | A route point |
| <code>monitor</code> | A monitor channel (not supported for Java) |
| <code>id</code> | A physical identifier |
| <code>voice</code> | A voice channel (only for Nortel Meridian switches, not supported for Java) |
| <code>agent</code> | A Meridian Link ACD position (only for Nortel Meridian switches) |
| <code>asai</code> | An Avaya DEFINITY device option (not supported for Java) |

Do not leave a space between the colon and *dn*. Enter the device type and *dn* as a single string, for example:

```
ctcTest> addmonitor route:231
```

You must specify a *dn* unless the device type is `monitor`. If the device type is a physical identifier (`id`), specify the number in hexadecimal.

If you do not specify a device type, CTC Test sets it to `dn` by default. The following command adds `dn 231` to the monitor channel:

```
ctcTest> addmonitor 231
```

addphysicalidmonitor

Environment: Java only

Syntax: **addphysicalidmonitor** *physicalId*

Adds a physical identifier to a monitor channel for monitoring. Specify *physicalId* in hexadecimal.

addroutelister

Environment: Java only

Syntax: **addroutelister**

Sets monitoring on and obtains information on activity on the assigned route point.

addroutemonitor

Environment: Java only

Syntax: **addroutemonitor** *dn*

Adds a route point to a monitor channel for monitoring. Specify the number for the route point.

answercall

Environment: C and Java

Syntax: **answercall** [*callrefid*]

Answers a call on a hands-free set.

The parameter *callrefid* is the reference identifier of the call. This is automatically included, or you can specify it (in hexadecimal).

asaidropparty

Environment: C and Java

Syntax: **asaidropparty** *partyid*

Disconnects the specified party from the call (only for Avaya DEFINITY).

partyid is the party identifier for the party to be dropped from the call.

To use this command, you must assign a channel to the `asaicall` device type using the `assign`, `secassign` or `create` command.

asaigetacdstatus

Environment: C and Java

Syntax: **asaigetacdstatus**

Returns ACD split information (only for Avaya DEFINITY).

To use this command, you must assign a channel to the `asai` device type using the `assign`, `secassign` or `create` command.

asaigetglobalrefid

Environment: C and Java

Syntax: **asaigetglobalrefid** [*callrefid*]

Returns the global reference identifier for a call (only for Avaya DEFINITY).

To use this command, you must assign a channel to the *asai* device type using the *assign*, *secassign* or *create* command.

The parameter *callrefid* is the reference identifier of the call. This is automatically included, or you can specify it (in hexadecimal).

asaiselectivelisten

Environment: C and Java

Syntax: **asaiselectivelisten** {on|off} *partyId* [*partyId1*,
partyId2 ... *partyId5*]

Temporarily disconnects a party from a call so that the party can no longer hear one or more of the other parties on the call. This command also reconnects the specified party. Only for Avaya DEFINITY.

Specify the following parameters:

on or *off* *on* sets the selective listen feature on. *off* sets the selective listen feature off.

partyid The party identifier for the listener. The listener is the party temporarily disconnected from one or more talkers on the call.

partyId1,partyId2
... *partyId5* Optional. Up to five party identifier(s) for the talkers the listener cannot hear. If you do not specify any party identifier for a talker, the listener is disconnected from all talkers on the call

To use this command, you must assign a channel to the *asaicall* device type using the *assign*, *secassign* or *create* command.

assign

Environment: C and Java

Syntax: [*n*] **assign** [{*dn*|*route*|*monitor*|*id*|*voice*|*specific*
|*agent*|*asai*|*asaicall*}:]*dn node log_id trans*
[*username password*][*url port-number*]

Assigns a communications channel.

Optionally, *n* is a number you can use to label the channel. If you do not specify a label number, CTC Test defaults it to 1.

Indicate the type of device to which you want to assign a channel by entering one of the following (if you do not specify a device type, CTC Test defaults to *dn*):

| | |
|-----------------|---|
| <i>dn</i> | A telephone or other telephony device (default) |
| <i>route</i> | A route point |
| <i>monitor</i> | A monitor channel |
| <i>id</i> | A physical identifier |
| <i>voice</i> | A voice channel (only for Nortel Meridian switches) |
| <i>specific</i> | A Meridian Link Regular DN (only for Nortel Meridian switches. Not supported for Java) |
| <i>agent</i> | A Meridian Link ACD position (only for Nortel Meridian switches) |
| <i>asai</i> | An ASAI-specific device for which you want to use ASAI API extensions (only for Avaya DEFINITY) |
| <i>asaicall</i> | A call for which you want to use ASAI API commands (only for Avaya DEFINITY) |

NOTE: Some CTC Test commands require you to specify a certain device type when you assign a channel. If this is required, it is indicated in the description of the CTC Test command.

Do not leave a space between the colon and the *dn* but enter the device type and *dn* as a single string. (Note that the colon is required even if the device type is a monitor channel.)

Specify the following parameters:

| | |
|-----------------|--|
| <i>dn</i> | The device number (note that this is not required if the device type is a monitor channel). If the device type is a physical identifier (<i>id</i>), the number must be in hexadecimal. Do not leave a space between the colon and the <i>dn</i> but enter the device type and <i>dn</i> as a single string. For example, <i>dn:1234</i> . Note that the colon is required even if the device type is a monitor channel. |
| <i>node</i> | The name of the call processing server. |
| <i>log_id</i> | The logical identifier of the link between the call processing server and the switch. |
| <i>trans</i> | Transport protocol for the connection. Note: do not use this parameter on Java clients. |
| <i>username</i> | Optional. A user name that has been configured for an authorized user. Refer to the <i>Intel NetMerge Call Processing Software Installation and Configuration Guide</i> for information about authorized users. Note: Only supported on clients using the Java API. Clients using the C API should use the secassign command to enter a user name. |

| | |
|-------------|--|
| password | Optional. A password that has been configured for an authorized user. Refer to the <i>Intel NetMerge Call Processing Software Installation and Configuration Guide</i> for information about authorized users. Note: Only supported on clients using the Java API. Clients using the C API should use the secassign command to enter a password. |
| url | Optional. The URL of the Web server used to reach the call processing server. Refer to the <i>Intel NetMerge Call Processing Software Installation and Configuration Guide</i> for more information. Only supported on clients using the Java API, typically external Java clients. |
| port-number | Optional. The port number used to communicate with the Web server specified by <i>url</i> . The default is 80. Only supported on clients using the Java API, typically external Java clients. |

In the following example, you allocate a label number of 4 to the channel assigned to device number 231 where the name for the call processing server is system1, the logical identifier is ctc_csta, and the transport protocol is TCP/IP:

```
ctcTest> 4 assign dn:231 system1 ctc_csta ncacn_ip_tcp
```

In the following example, you assign a monitor channel:

```
ctcTest> 2 assign monitor: ctcserv ctc_ml ncacn_ip_tcp
```

associatedata

Environment: C and Java

Syntax: **associatedata** *callrefid data*

Associates data with a call and passes it to the switch.

Specify the following parameters:

| | |
|-----------|--|
| callrefid | The callrefid of the call (in hexadecimal). |
| data | The data you want to associate with the call. This is a string up to 127 characters in length. If you leave this blank, you will delete any server-based application data. |

Not all switches support application data, but you can configure the call processing server to store application data instead of the switch.

associateremotedata

Environment: C and Java

Syntax: **associateremotedata** *calleddn remoteserver
remotelogid [data] [url]*

Associates data with a device at a remote switch connected to a remote call processing server. Application data passed in this way is also known as distributed data.

Specify the following parameters:

| | |
|---------------------|--|
| calleddn | The number of the device on the remote switch that will receive the call. The number that you specify must be the same as that used to redirect or route the call from the local switch to the remote switch. |
| remoteserver | The network name or address of the remote call processing server. |
| remotelogid | The logical identifier of the link between the remote call processing server and the remote switch. |
| data | Optional. The data to associate with the call. This is a string up to 127 characters in length. If you leave this blank, you will delete any server-based application data. |
| url | Optional. The URL of the Web server used to reach the remote call processing server. Note that if you want to use this, both the local and the remote call processing server must have the Java RMI interface enabled. Typically, you would supply the URL from clients using the Java API, but it is also supported for clients using the C API. |

For more information about using application data and distributed data, refer to the *Intel NetMerge Call Processing Software Installation and Configuration Guide*.

cancelcall

Environment: C and Java

Syntax: **cancelcall** [*callrefid*]

Disconnects a consultation call.

The callrefid of the call is automatically included or you can specify a callrefid (in hexadecimal).

conferencejoin

Environment: C and Java

Syntax: **conferencejoin** [*heldrefid* [*callrefid*]]

Joins participants into a conference call.

The heldrefid and callrefid of the calls are automatically included or you can specify the heldrefid and callrefid (in hexadecimal). If you specify a callrefid, you must specify the heldrefid.

consultationcall

Environment: C and Java

Syntax: **consultationcall** {*trans*|*conf*} *dn* [*data*]

Places a consultation call to another device.

Specify:

- One of the following:

trans Initiates a consultation call for a transfer

conf Initiates a consultation call for a conference

- The DN of the target device (*dn*)
- (Optional). A string of up to 127 characters (*data*)

create

Environment: Java only

Syntax: `[n] create [{device|route|monitor|id|agent|mlp|asai|asaicall}]:dn node log_id [username password][url port-number]`

Creates a communications channel.

Optionally, *n* is a number you can use to label the channel. If you do not specify a label number, CTC Test uses the default of 1.

Indicate the type of device to which you want to assign a channel by entering one of the device types listed below. If you do not indicate a device type, CTC Test defaults to device:

| | |
|----------|---|
| device | A telephone or other telephony device (default) |
| route | A route point |
| monitor | A monitor channel |
| id | A physical identifier |
| agent | A Meridian Link ACD position (only for Nortel Meridian switches) |
| mlp | A voice channel (only for Nortel Meridian switches) |
| asaicall | A call for which you want to use ASAI API commands (only for Avaya DEFINITY) |
| asaicall | An ASAI-specific device for which you want to use ASAI API extensions (only for Avaya DEFINITY) |

NOTE: Some CTC Test commands require you to specify a certain device type when you assign a channel. If this is required, it is indicated in the description of the command.

Specify the following parameters after the device type:

| | |
|----------|---|
| dn | The device number (this is not required if the device type is a monitor channel). If the device type is <i>id</i> , specify the physical identifier in hexadecimal. Do not leave a space between the colon and <i>dn</i> . Enter the device type and <i>dn</i> as a single string. (Note that the colon is required even if the device type is a monitor channel.) |
| node | The name of the call processing server. |
| log_id | The logical identifier of the link between the call processing server and the switch. |
| username | Optional. A user name that has been configured for an authorized user. Refer to the <i>Intel NetMerge Call Processing Software Installation and Configuration Guide</i> for information about authorized users. |


```

[/ErcAccountCode=...]
[/ErcAuthCode=...]
[/ErcCancelCallback=...]
[/ErcEnterDTMF=...]
[/ErcFreeQueuePos=...]
[/ErcFwdACDGroup=...]
[/ErcKeepQueuePos=...]
[/ErcMessageDiversion=...]
[/ErcPressProgKey=...]
[/ErcSetAssociateData=...]
[/HcmRejectCall=...]
[/HcmRouteTrigger=...]
[/RawData=...]
[/RBMDData=...]

```

Enables the exchange of private data with the switch (only for CSTA switches).

For the full syntax of each of the `escape` parameters, see below. Only use parameters that are appropriate to your type of switch, as follows:

| | |
|--|--|
| Parameter beginning <code>/RBMDData</code> | Any CSTA Phase I, Phase II or Phase III switch |
| Parameter beginning <code>/RawData</code> | Any CSTA Phase II or Phase III switch |
| Parameters beginning <code>/Alc</code> | Alcatel 4400 (CSTA Phase II or Phase III) |
| Parameters beginning <code>/Erc</code> | Ericsson† MD110 BC9 or later (CSTA Phase I) |
| Parameters beginning <code>/Hcm</code> | Siemens† Hicom† 300E (CSTA Phase I) |

There may also be switch-based functional restrictions that further limit the exchange of private data. Please refer to your switch manufacturer for definitive information.

Please note these points:

- The descriptions in this section use mixed upper and lowercase to clarify the meaning of the parameters. However, you can still enter them in lowercase. An exception to this is after the equals sign (=), where you must enter all keywords in the case shown. For example, enter `On` and not `on`.
- Where you are entering more than one parameter, separate each with a comma (as shown).
- Do not leave blank spaces.

Specify at least one of the following, typing in the content indicated after the equals sign.

`/AlcDateAndTime=` `dateandtime`

Sets the date and time on the switch.
Enter the date and time in the format YYYYMMDDHHMMSS.T (for example, 19970203123015.0).

`/AlcOther=` `identifier,data`

Sends other, undefined data to the switch.
Specify the following:

`identifier` Two hexadecimal digits (for example, 7a)

`data` Hexadecimal digit string (for example, 1256780a)

`/AlcServiceOptions=` `{c|h|p|a}`

Selects an Alcatel service option.
Specify one or more of the following:

`c` Call Progress Tone inhibition

`h` Hold Tone inhibition

`p` Priority Transfer

`a` Auto Originate

`/ErcAccountCode=` `accountcode[,callrefid]`

Specifies the account code (up to 11 characters long).
The callrefid of the call is automatically included or you can specify a callrefid (in hexadecimal).

`/ErcAuthCode=` `authcode`

Specifies the authorization code (up to 11 characters long).

`/ErcCancelCallbFack=` `[dn]`

Cancels call back on a device.
Optionally, specify the DN (up to 12 digits long).

`/ErcEnterDTMF=` `dtmfdigits[,callrefid]`

Specifies DTMF digits to send (up to 12 digits long).
The callrefid of the call is automatically included or you can specify a callrefid (in hexadecimal).

`/ErcFreeQueuePos=` `deviceid,callrefid`

Releases the queue position for a deflected call.
Specify the following:

`deviceid` The device DN (up to 12 digits long)

`callrefid` The callrefid of the call (in hexadecimal)

`/ErcFwdACDGroup=` `{On|Off},acddevice,fdtodevice`

Forwards an ACD group to another destination.
Specify the following:

`{On|Off}` Turn forwarding on or off

`acddevice` The ACD group number (up to 12 digits long)

`fdtodevice` The destination DN (up to 12 digits long)

`/ErcMessageDiversion=` `deviceid,{On|Off},diversiontype[,timeordate]`

Sets diversion to an operator.
Specify the following:

`deviceid` The destination DN (up to 12 digits)

`On|Off` Turn diversion on or off

`diversiontype` The diversion type (a digit in the range 0 – 9)

Optionally, you can add time or date information (up to 5 characters).

`/ErcPressProgKey=` `keynumber`

Presses user key numbers on an MD110 phone.
Specify the key number string (up to four characters).

`/ErcSetAssociateData=` `appdata[,callrefid]`

Associates a dialable string with an external caller.
Specify the string (up to 21 characters long).
The callrefid of the call is automatically included or you can specify a callrefid (in hexadecimal).

`/HcmRejectCall=` `routeref`

Rejects a route call request.
Specify the route reference identifier string (four hexadecimal digits).

`/HcmRouteTrigger=` `{On|Off}[,triggerobject]`

Triggers all configured Route Control Groups (RCGs).
Optionally, you can specify the DN to be triggered (up to 24 digits long).

`/RawData=` `data`

Sends a raw hexadecimal data string (for example, 1256780a).

`/RBMDData=` `manid,data`

Sends a manufacturer identifier and a raw hexadecimal data string.
Specify the following:

`manid` The manufacturer identification string (for example,
1.2.1200.12)

`data` The hexadecimal data string (for example,
1256780a)

exit

Environment: C and Java

Syntax: **exit**

Exits the CTC Test program.

getagentstatus

Environment: C and Java

Syntax: **getagentstatus** [`/data=...`]

Obtains the operating mode of the ACD agent.

Use the `/data` qualifier to include optional data of up to 127 characters
(supported for Avaya DEFINITY switches only). For example, an agent password.

getallagentsstates

Environment: C

Syntax: **getallagentsstates** [`/group=agentgroup`]

You can use the optional `group` qualifier to specify the name of the agent group.

For a channel assigned to an agent, specify this parameter to return the agent's state associated with a specific group. If you omit this parameter, the command returns the agent's states for all associated groups.

For a channel assigned to an ACD number, specify this parameter to return the states for up to 32 agents at that ACD. If you omit this parameter, the command returns all agent states for all groups at the ACD number.

getallcallforwards

Environment: C and Java

Syntax: **getallcallforwards**

Obtains all of the call forward settings for the assigned device.

Supported for CSTA switches only.

getauditoryapparatusinfo

Environment: C and Java

Syntax: **getauditoryapparatusinfo** *apparatusid*

Returns information about the apparatus at the assigned device.

For *apparatusid*, specify the identifier for the apparatus on the assigned device.

getautoanswer

Environment: C and Java

Syntax: **getautoanswer**

Returns the Auto Answer setting at the assigned device.

getbuttoninfo

Environment: C and Java

Syntax: **getbuttoninfo** *buttonid*

Returns information about the specified button at the assigned device.

For *buttonid*, specify the identifier for the button on the assigned device.

getcallforward

Environment: C and Java

Syntax: `getcallforward`

Obtains the call forward setting for incoming calls on the assigned device.

getchannelinformation

Environment: C and Java

Syntax: `getchannelinformation`

Gets data on the communications channel and the device to which it is assigned.

getdisplayinfo

Environment: C and Java

Syntax: `getdisplayinfo displayid`

Returns information about the display at the assigned device.

For *displayid*, enter the identifier for the display at the assigned device. Specify the identifier in hexadecimal.

getdonotdisturb

Environment: C and Java

Syntax: `getdonotdisturb`

Obtains the do-not-disturb setting on the assigned device.

getevent

Environment: C only

Syntax: `getevent [poll]`

Obtains information on activity on a device.

Include `poll` if you want to create a synchronous version of the `getevent` command. After you issue a `getevent poll` command, CTC Test will wait until an event occurs on the assigned device before allowing you to continue.

Do not include `poll` if you want to have information continue to return until:

- You turn monitoring off.
- A status other than `ctcSuccess` or `ctcEventDataLost` is returned.

gethookswitchstatus

Environment: C and Java

Syntax: `gethookswitchstatus hookswitchid`

Returns information about the hookswitch status (on or off) at the assigned device.

hookswitchid is the identifier for the hookswitch on the assigned device. Specify the identifier in hexadecimal.

getlampinfo

Environment: C and Java

Syntax: `getlampinfo lampid`

Returns information about the specified lamp on the assigned device.

lampid is the identifier for the lamp at the assigned device.

getlampmode

Environment: C and Java

Syntax: `getlampmode lampid`

Returns characteristics of the specified lamp at the assigned device, for example, its color and brightness.

lampid is the identifier for the lamp at the assigned device.

getmessagewaiting

Environment: C and Java

Syntax: `getmessagewaiting`

Obtains the message waiting indicator setting on the assigned device.

getmicrophonegain

Environment: C and Java

Syntax: `getmicrophonegain apparatusid`

Returns the gain setting (input level) for the apparatus microphone at the assigned device.

For *apparatusid*, enter the identifier for the apparatus at the assigned device.

getmicrophonemute

Environment: C and Java

Syntax: `getmicrophonemute apparatusid`

Returns the mute setting (on or off) for the apparatus microphone at the assigned device.

For *apparatusid*, enter the identifier for the apparatus at the assigned device.

getmonitor

Environment: C and Java

Syntax: `getmonitor`

Obtains the monitoring state for the assigned device.

getprivatedata

Environment: C and Java

Syntax: `getprivatedata`

Retrieves private data. Supported for CSTA switches only.

getprivateeventdata

Environment: C and Java

Syntax: `getprivateeventdata`

Retrieves private data associated with the last event returned on a channel. Supported for CSTA switches only.

getprivateroutedata

Environment: C and Java

Syntax: **getprivateroutedata**

Retrieves private data associated with the last route request received on a channel. Supported for CSTA switches only.

getringerstatus

Environment: C and Java

Syntax: **getringerstatus** *ringerid*

Returns information about the ringer at the assigned device. This can include ringer mode (ringing or not ringing), ring count, ring pattern and ring volume.

For *ringerid*, specify the identifier for the ringer at the assigned device.

getroutequery

Environment: C only

Syntax: **getroutequery** [*dn* [*poll*]]

Returns a route request for a call when it reaches a route point. To use this command, you must assign a channel to the *route* device type.

If you wish, you may:

- Specify a destination DN (*dn*) for the call to automatically route it.
- Include *poll* if you want the command executed synchronously. This means that you must wait for the command to return with a route request. If you specify *poll*, you must also include *dn* on the command line.

Do not include *poll* if you want the command executed asynchronously.

Note that *getroutequery* only gets one route query at a time. If you call *respondtoroutequery*, you will have to call *getroutequery* again for the next call you are interested in.

getroutingenable

Environment: C and Java

Syntax: **getroutingenable**

Obtains routing enabled status. To use this command, you must assign a channel to the `route` device type.

If routing is enabled, the switch passes route requests to the Call Processing Software when it receives a call for the route point. If disabled, the switch does not pass route requests to the Call Processing Software for the route point. Use `setroutingenable` to set the routing state.

getspeakermute

Environment: C and Java

Syntax: **getspeakermute** *apparatusid*

Returns information about the speaker mute setting at the assigned device.

For *apparatusid*, specify the identifier for the apparatus on the assigned device.

getspeakervolume

Environment: C and Java

Syntax: **getspeakervolume** *apparatusid*

Returns information about the speaker volume setting at the assigned device.

For *apparatusid*, specify the identifier for the apparatus on the assigned device.

hammer

Environment: C and Java

Syntax: **hammer** *dn iterations*

Loops `makecall` and `hangupcall` to a telephone.

Specify the target DN (*dn*) and the number of iterations of the test you want performed.

hangupcall

Environment: C and Java

Syntax: **hangupcall** [*callrefid*]

Clears a call on a device and returns it to the null state.

The call reference id (*callrefid*) of the call is automatically included or you can specify a *callrefid* (in hexadecimal).

help

Environment: C and Java

Syntax: **help** [*command*]

Displays help text for a command. If you do not enter **help** without a command, you see the Table of Contents for CTC Test help. Double-click the book icon to see a complete list of CTC Test commands.

Enter a specific command name to get help on that command.

holdcall

Environment: C and Java

Syntax: **holdcall** [*callrefid*]

Puts a call on hold.

The call reference id (*callrefid*) of the call is automatically included, or you can specify a *callrefid* (in hexadecimal).

makecall

Environment: C and Java

Syntax: **makecall** *dn* [*data*]

Makes a call.

Specify the DN (*dn*) of the call destination. Optionally, you can pass a string of up to 127 characters (*data*).

makepredictivecall

Environment: C and Java

Syntax: **makepredictivecall** *dn* {def|act|del|drop|conn}
[*data* [*rings*]]

Allows a virtual party on a switch to initiate calls.

Specify the DN (*dn*) of the destination device, and one of the following:

| | |
|-------------|--|
| <i>def</i> | Enable the switch default processing of the call |
| <i>act</i> | Call is successful when called device answers |
| <i>del</i> | Call is successful when put through to called device |
| <i>drop</i> | End call if answer machine detected |
| <i>conn</i> | Continue call if answer machine detected |

Optionally, you can pass the following parameters:

| | |
|--------------|--|
| <i>data</i> | A string of up to 127 characters |
| <i>rings</i> | The number of rings made at the called device before the call fails. Specify 0 to use the switch default (15) or specify a value between 2 and 15. (Avaya DEFINITY only.) If you specify the <i>rings</i> parameter, you must also specify a value for the preceding <i>data</i> parameter. |

mlpclosevoicefile

Environment: C and Java

Syntax: **mlpclosevoicefile**

Closes an open voice file on a Meridian Mail system (only for Nortel Meridian switches).

To use this command, you must first use the `assign` or `secassign` command to assign a channel to the `voice` device type (C only) or `create` a channel to the `mlp` device type (Java only).

mlpcollectdigits

Environment: C and Java

Syntax: **mlpcollectdigits** [*numofdigits* [{on|off}] [*timeout* [*termkey*]]]

Collects DTMF digits entered by a user (only for Nortel Meridian switches).

Optionally, you can specify the following:

| | |
|--------------------|--|
| <i>numofdigits</i> | Number of digits to be collected (default = 4). |
| on off | Whether the key buffer is cleared or not (default = on). If you specify an on/off value, you must include the preceding <i>numofdigits</i> parameter. |
| <i>timeout</i> | Number of seconds allowed between digits (default = 5). If you specify a <i>timeout</i> value, you must include the preceding <i>numofdigits</i> and on/off parameters. |
| <i>termkey</i> | Character used to indicate the end of dialing (default = #). If you specify a <i>termkey</i> value, you must include the preceding <i>numofdigits</i> , on/off, and <i>timeout</i> parameters. |

To use this command, you must first use the `assign` or `secassign` command to assign a channel to the `voice` device type (C only) or `create` a channel to the `mlp` device type (Java only).

mlplogoffmailbox

Environment: C and Java

Syntax: **mlplogoffmailbox**

Logs off from a Meridian Mail account (only for Nortel Meridian switches).

To use this command, you must first use the `assign` or `secassign` command to assign a channel to the `voice` device type (C only) or `create` a channel to the `mlp` device type (Java only).

mlplogonmailbox

Environment: C and Java

Syntax: **mlplogonmailbox** *userid password*

Logs on to a Meridian Mail account (only for Nortel Meridian switches).

Specify the DN for the Meridian Mail system as defined on the switch and the password for access to the Meridian Mail system.

To use this command, you must first use the `assign` or `secassign` command to assign a channel to the `voice` device type (C only) or `create` a channel to the `mlp` device type (Java only).

mlpopenvoicefile

Environment: C and Java

Syntax: **mlpopenvoicefile** *filename*

Opens a voice file on a Meridian Mail system (only for Nortel Meridian switches).

Specify the voice file filename.

To use this command, you must first use the `assign` or `secassign` command to assign a channel to the `voice` device type (C only) or `create` a channel to the `mlp` device type (Java only).

mlpplaymessage

Environment: C and Java

Syntax: **mlpplaymessage** [{on|off} [{on|off} [/segment=*n,m,...*]]]

Plays a message to a caller (only for Nortel Meridian switches).

Optionally, you can specify the following:

| | |
|------------------------|--|
| <code>on off</code> | Whether the key buffer is cleared before playing or not (default = on). |
| <code>on off</code> | Whether the message can be interrupted by pressing a key or not (default = on). If you include a value for this parameter, you must include an on/off value for the preceding parameter. |
| <code>/segment=</code> | Starting offset of the voice segment(s) to be played from the file. If you include this parameter on the command line, you must include on/off values for both preceding parameters. |

To use this command, you must first use the `assign` or `secassign` command to assign a channel to the voice device type (C only) or create a channel to the `mlp` device type (Java only).

parkcall

Environment: C and Java

Syntax: `parkcall callrefid parkdn [data]`

Moves a call from a telephony device to a parking lot that is defined on the switch.

For the *callrefid*, specify the identifier for the call you want to park. For the *parkdn*, specify the number for the parking lot. Optionally, you can supply a string of up to 127 characters for *data*.

pickupcall

Environment: C and Java

Syntax: `pickupcall [dn [callrefid]]`

Answers a call which is ringing at another device.

If the ringing telephone is in the same pickup group, you do not need to provide its DN (*dn*). If the ringing telephone is not in the same pickup group, you do need to provide its DN.

The call reference id (*callrefid*) of the call is automatically included or you can specify *callrefid* (in hexadecimal). If you specify a *callrefid*, you must also include a *dn* value on the command line.

pressbutton

Environment: C and Java

Syntax: `pressbutton buttonid`

Emulates pressing a button at the assigned device.

For the *buttonid*, specify the identifier for the button on the assigned device.

quit

Environment: C only

Syntax: `quit`

Quits the CTC Test program. This performs the same function as `exit`.

reconnectcall

Environment: C and Java

Syntax: `reconnectcall [heldrefid [callrefid]]`

Disconnects a consultation call and reconnects a held call.

Both the *heldrefid* and the *callrefid* are automatically included. Alternatively, you can specify either or both:

- For *heldrefid*, enter the call reference identifier for the held call, in hexadecimal.
- For *callrefid*, enter the call reference identifier for the active call being disconnected, in hexadecimal.

Note that if you specify *callrefid*, you must also specify *heldrefid*.

removeagentdevicemonitor

Environment: Java only

Syntax: `removeagentdevicemonitor dn`

Removes an agent position from a monitor channel (only for Nortel Meridian switches). For *dn*, specify the agent position you no longer want to monitor.

removedevicemonitor

Environment: Java only

Syntax: `removedevicemonitor dn`

Removes a device from a monitor channel so that it is no longer monitored on that channel. For *dn*, specify the number for the device you no longer want to monitor.

removeeventlistener

Environment: Java only

Syntax: **removeeventlistener**

Sets monitoring off.

removemonitor

Environment: C and Java

Syntax: **removemonitor**
[{dn|route|monitor|id|voice|agent} :]dn

Removes a device from monitoring on a monitor channel.

Enter one of the following to indicate the type of device:

| | |
|---------|--|
| dn | A telephone or other telephony device (default if no device type is specified) |
| route | A route point |
| monitor | A monitor channel (not supported for Java) |
| id | A physical identifier |
| voice | A voice channel (only for Nortel Meridian switches, not supported for Java) |
| agent | A Meridian Link ACD position (only for Nortel Meridian switches) |

Do not leave a space between the colon and the *dn* but enter the device type and *dn* as a single string, for example:

```
ctcTest> removemonitor route:231
```

You must specify a *dn* unless the device type is `monitor`. For example:

```
ctcTest> removemonitor monitor:
```

If the device type is `id`, specify the physical identifier in hexadecimal.

If you do not specify a device type, CTC Test sets it to `dn` by default. The following command removes `dn 231` from the monitor channel:

```
ctcTest> removemonitor 231
```

removephysicalidmonitor

Environment: Java only

Syntax: **removephysicalidmonitor** *physicalIdentifier*

Removes a physical identifier from monitoring on a monitor channel.

Specify the physical identifier in hexadecimal.

removeroutemonitor

Environment: Java only

Syntax: **removeroutemonitor** *dn*

Removes a route point from monitoring on a monitor channel.

respondtoinactivecall

Environment: C and Java

Syntax: **respondtoinactivecall** {*camp*|*barge*|*ring*}
[*callrefid*]

Sets a response if there is an existing call on the destination device.

Specify one of the following:

| | |
|--------------|---|
| <i>camp</i> | Wait until the existing call is cleared |
| <i>barge</i> | Barge in on the existing call |
| <i>ring</i> | Invoke ring back |

The call reference id (*callrefid*) of the call is automatically included or you can specify *callrefid* (in hexadecimal).

respondtoroutequery

Environment: C and Java

Syntax: **respondtoroutequery** *dn* [*route_id*] [*data*]

Provides a destination for a call in response to a route request.

Specify the DN (*dn*) of the new destination.

Optionally, you can specify the following:

| | |
|-----------------------|--|
| <code>route_id</code> | Route request identifier (defaults to the identifier from the current <code>getroutequery</code> return) |
| <code>data</code> | A string of up to 127 characters |

retrievecall

Environment: C and Java

Syntax: **retrievecall** [*callrefid*]

Retrieves a call that is on hold.

The call reference identifier (*callrefid*) of the call is automatically included; alternatively, you can specify *callrefid* (in hexadecimal).

routecall

Environment: Java

Syntax: **routecall** *dn* [*route_id*] [*data*]

Provides a destination for a call in response to a route request.

Specify the DN (*dn*) of the new destination.

Optionally, you can specify the following:

| | |
|-----------------------|---|
| <code>route_id</code> | Route request identifier (defaults to the identifier from the current <code>getroutequery</code> return) |
| <code>data</code> | A string of up to 127 characters. Note that if you pass <i>data</i> with this command, you must also pass a <i>route_id</i> . |

secassign

Environment: C only

Syntax:

The same as for the **assign** command. However, when you enter the **secassign** command, you are prompted for a user name and password.

Assigns a communications channel.

See the **assign** command for details of the syntax and parameters.

Note: This command is not supported for clients using the Java API.

senddtmf

Environment: C and Java

Syntax: **senddtmf** *digits*

Transmits DTMF tones.

Enter the keys you want transmitting as DTMF tones in the *digits* parameter.

setagentstatus

Environment: C and Java

Syntax: **setagentstatus**

```
{rdy|nrdy|busy|acw|ow|own|logi|logo|
logon|nrdyall|nrdyacd|nrdyidn|actcode|
callsup|emrg} [/data=../logi=../group=..]
```

Sets the status for an ACD agent.

Specify one of the following:

| | |
|-------------|--|
| <i>rdy</i> | Agent is ready to receive calls. |
| <i>nrdy</i> | Agent is not ready to receive calls. |
| <i>busy</i> | Agent is busy. |
| <i>acw</i> | Agent is completing details of a call (not supported by Nortel Meridian 1). |
| <i>ow</i> | Agent cannot take calls because of other work (not supported by Nortel Meridian 1). |
| <i>own</i> | Agent cannot take calls because of other work, where <i>n</i> is a number from 1 to 9 indicating a switch-specified reason code (only for Avaya DEFINITY supporting Expert Agent Selection). To use this command, you must assign the channel to the <i>asai</i> device type. |

| | |
|----------------------|---|
| <code>logi</code> | Agent is logging in. |
| <code>logo</code> | Agent is logging out. |
| <code>logon</code> | Agent is logging out, where <i>n</i> is a number from 1 to 9 indicating a switch-specified reason code (only for Avaya DEFINITY supporting Expert Agent Selection). To use this command, you must assign the channel to the <code>asai</code> device type. |
| <code>nrdyall</code> | Agent is not ready to receive calls (existing calls are not disconnected) (Nortel Meridian 1 only). |
| <code>nrdyacd</code> | Agent is not ready to receive calls (an existing ACD call is not disconnected) (Nortel Meridian 1 only). |
| <code>nrdyidn</code> | Agent is not ready to receive calls (an existing internal DN call is not disconnected) (Nortel Meridian 1 only). |
| <code>actcode</code> | Enters the activity codes on behalf of an agent set. Specify the activity code in the <code>/data=</code> parameter (Nortel Meridian 1 only). |
| <code>callsup</code> | Activates the Call Supervisor feature on behalf of an agent set (Nortel Meridian 1 only). |
| <code>emrg</code> | Activates the Emergency feature on behalf of an agent set. Agents can use this to connect to their supervisors (Nortel Meridian 1 only). |

Optionally, you can add these qualifiers:

| | |
|----------------------|---|
| <code>/data=</code> | Include optional data such as a password. If you specify <code>actcode</code> for <i>status</i> , enter the activity code here. |
| <code>/logi=</code> | Include optional data for logical agents (not supported by Nortel Meridian 1). |
| <code>/group=</code> | Include optional data for an agent group (not supported by Nortel Meridian 1). |

setautoanswer

Environment: C and Java

Syntax: `setautoanswer [on|off]`

Sets the Auto Answer feature on or off at the assigned device.

setbuttoninfo

Environment: C and Java

Syntax: `setbuttoninfo buttonid [/label=buttonlabel]
[/dn=buttonnumber]`

Sets information about the specified button at the assigned device.

For *buttonid*, specify the identifier for the button on the assigned device.

As an option, you can also specify the following:

/label=buttonlabel Specify the label you want to associate with the button.

/label=buttonnumber Specify the dialable number you want to associate with the button. For example, a speed-dial number.

setcallforward

Environment: C and Java

Syntax: **setcallforward** {all|b|dnd|eb|ednd|eim|ena|ib|idnd|iim|im|ina|na|nab} *dn*

Redirects incoming calls to another device.

Specify one of the following:

| | |
|------|---|
| all | All calls |
| b | All calls when the assigned telephone is busy |
| dnd | All calls when Do Not Disturb is set |
| eb | External calls when the assigned telephone is busy |
| ednd | External calls when Do Not Disturb is set |
| eim | External calls immediate |
| ena | External calls when there is no answer |
| ib | Internal calls when the assigned telephone is busy |
| idnd | Internal calls when Do Not Disturb is set |
| iim | Internal calls immediate |
| im | All calls immediate |
| ina | Internal calls when there is no answer |
| na | All calls when there is no answer (not supported for Java) |
| nab | All calls when there is no answer and when the assigned telephone is busy |

For *dn*, specify the DN of the other device.

setcallmonitor

Environment: C and Java

Syntax: **setcallmonitor** {on|off} *callrefid*

Sets monitoring on or off for channels assigned to device type asaicall.

For *callrefid*, specify the call reference identifier of the call, in hexadecimal.

setdisplayinfo

Environment: C and Java

Syntax: **setdisplayinfo** *displayid displaytext*
[*/pbrow=physicalbaserow*]
[*/pbcoll=physicalbasecolumn*]
[*/offset=offsetnumber*]

Sets the contents of a physical display at the assigned device.

Specify the following parameters:

| | |
|-----------------------------------|---|
| <i>displayid</i> | The identifier for the display at the assigned device. Specify the identifier in hexadecimal. If there is only one display at the assigned device, enter 0 (zero) for the identifier. |
| <i>displaytext</i> | The text you want to show on the display at the assigned device. |
| <i>/pbrow=physicalbaserow</i> | Optional. Specify the number for the logical row that appears as the first row on the physical display. |
| <i>/pbcoll=physicalbasecolumn</i> | Optional. Specify the number for the logical column that appears as the first column on the physical display. |
| <i>/offset=offsetnumber</i> | Optional. Specify the offset on the logical display where text is to start. |

setdonotdisturb

Environment: C and Java

Syntax: **setdonotdisturb** {on|off}

Sets do-not-disturb on or off.

sethookswitchstatus

Environment: C and Java

Syntax: **sethookswitchstatus** *hookswitchid* {on|off}

Sets the hookswitch state (on or off) at the assigned device.

For *hookswitchid*, enter the identifier for the hookswitch on the assigned device. Specify the identifier in hexadecimal.

Specify *on* or *off* to set the state.

setlampmode

Environment: C and Java

Syntax: **setlampmode** *lampid* *lampmode* [/col=*color*]
[/*bright=brightness*]

Sets the mode and features of a lamp at the assigned device.

Specify the following parameters:

lampid The identifier for a lamp at the assigned device. Specify the identifier in hexadecimal.

lampmode The lamp mode you want to set. Specify one of the following:

- brokenflutter
- flutter
- off
- steady
- wink

Alternatively, you can specify a value between 6 and 100 that identifies a switch-specific mode.

/col=color Optional. *color* is the color for the lamp. Specify one of the following:

- none
- red
- yellow
- green
- blue

Alternatively, you can specify a value between 6 and 100 that identifies a switch-specific mode.

/bright=brightness Optional. *brightness* is the lamp brightness. Specify one of the following:

- normal
- dim
- bright

setmessagewaiting

Environment: C and Java

Syntax: **setmessagewaiting** {on|off} [/dn=*numbertodial*]

Specify the following parameters:

{on/off} Sets the message waiting indicator on or off.

/dn=numbertodial Optionally, specify the number you dial to access messages.

setmicrophonegain

Environment: C and Java

Syntax: **setmicrophonegain** *apparatusid* *gain*

Sets the gain (input level) for a microphone associated with the auditory apparatus at the assigned device.

Specify the following parameters:

apparatusid The identifier for the apparatus on the assigned device. Specify the identifier in hexadecimal.

gain Specify one of the following for *gain*:

- minimum
- maximum
- incremented
- decremented

Alternatively, specify a value between 1 and 99 that identifies an input level that falls between silence and maximum gain. The granularity of the values 1 through 99 is device specific.

setmicrophonemute

Environment: C and Java

Syntax: **setmicrophonemute** *apparatusid* {on|off}

Sets the microphone mute on the associated apparatus at the assigned device.

Specify the following parameters:

apparatusid The identifier for the apparatus on the assigned device. Specify the identifier in hexadecimal.

{on|off} The state you want to set. Specify on or off.

setmonitor

Environment: C and Java

Syntax: **setmonitor** {on|off}

Sets monitoring on or off.

setprivatedata

Environment: C and Java

Syntax: **setprivatedata** [/AlcDateAndTime=...]
 [/AlcOther=...]
 [/AlcServiceOptions=...]
 [/ErcKeepQueuePos=...]
 [/HcmAssociateData=...]
 [/HcmAutoAnswerMode=...]
 [/RawData=...]
 [/RBMDData=...]

Send private data to the switch (only for CSTA switches).

For the full syntax of each of the **setprivatedata** parameters, see below.

Only use parameters that are appropriate to your type of switch:

| | |
|-------------------------------|--|
| Parameter beginning /RBMDData | Any CSTA Phase I, Phase II or Phase III switch |
| Parameter beginning /RawData | Any CSTA Phase II or Phase III switch |
| Parameters beginning /Alc | Alcatel 4400 (CSTA Phase II or Phase III) |
| Parameters beginning /Erc | Ericsson† MD110 BC9 or later (CSTA Phase I) |

There may also be switch-based functional restrictions that further limit the exchange of private data. Please refer to your switch manufacturer for definitive information.

Please note these points:

- The descriptions in this section use mixed upper and lowercase to clarify the meaning of the parameters. However, you can still enter them in lowercase. An exception to this is after the equals sign (=), where you must enter all keywords in the case shown. For example, enter On and not on.
- Where you are entering more than one parameter, separate each with a comma (as shown).
- Do not leave blank spaces.

Specify at least one of the following, typing in the content indicated after the equals sign.

| | |
|---------------------|--|
| /AlcDateAndTime= | dateandtime |
| | Sets the date and time on the switch. Enter the date and time in the format YYYYMMDDHHMMSS.T (for example, 19970203123015.0). |
| /AlcOther= | identifier,data |
| | Sends other, undefined data to the switch. Specify the following: |
| | identifier Two hexadecimal digits (for example, 7a) |
| | data Hexadecimal digit string (for example, 1256780a) |
| /AlcServiceOptions= | {c h p a} |
| | Selects an Alcatel service option. Specify one of the following: |
| | c Call Progress Tone inhibition |
| | h Hold Tone inhibition |
| | p Priority Transfer |
| | a Auto Originate |
| /ErcKeepQueuePos= | |
| | Holds the queue position for a call that has been deflected temporarily. Note that the = sign is required even though there is nothing further on the command line. |

| | | |
|----------------------------------|-------------------------|--|
| <code>/HcmAssociateData=</code> | <code>appdata</code> | Specifies an application data string (up to 31 characters long). |
| <code>/HcmAutoAnswerMode=</code> | <code>{On Off}</code> | Specifies the AutoAnswer setting for the originating device. If you specify On, AutoAnswer is attempted. If you specify Off, AutoAnswer is disabled. |
| <code>/RawData=</code> | <code>data</code> | Sends a raw hexadecimal data string (for example, 1256780a). |
| <code>/RBMDData=</code> | <code>manid,data</code> | Sends a manufacturer identifier and a raw hexadecimal data string. Specify the following: |
| | <code>manid</code> | The manufacturer identification string (for example, 1.2.1200.12) |
| | <code>data</code> | The hexadecimal data string (for example, 1256780a) |

setringerstatus

Environment: C and Java

Syntax: **set** `ringerstatus` *ringerid*
`/mode={ringing|notringing}`
`[/pattern=patternnumber]`
`[/volume=ringvolume]`

Sets options for the specified ringer at the assigned device.

Specify the following parameters:

| | |
|-----------------------------------|---|
| <i>ringerid</i> | The identifier for the ringer you want to set. Specify the identifier in hexadecimal. |
| <i>/mode=ringing notringing</i> | Specify either ringing or notringing. |
| <i>/pattern=patternnumber</i> | Optional. The number for the ring pattern (<i>patternnumber</i>). Ring patterns and the numbering for supported ring patterns are specific to the switch. |
| <i>/volume=ringvolume</i> | Optional. Specify one of the following for <i>ringvolume</i> : <ul style="list-style-type: none">• minimum• maximum• incremented• decremented Alternatively, specify a value between 1 and 99 for a level of volume that falls between silence and maximum volume. The granularity of the values 1 through 99 and the relationship between volume and loudness are both device specific. |

setroutingenable

Environment: C and Java

Syntax: **setroutingenable** {on|off}

Sets routing for an assigned route point on or off.

setspeakermute

Environment: C and Java

Syntax: **setspeakermute** *apparatusid* {on|off}

Sets the speaker mute on the associated apparatus at the assigned device.

Specify the following parameters:

| | |
|--------------------|---|
| <i>apparatusid</i> | The identifier for the apparatus on the assigned device. Specify the identifier in hexadecimal. |
| {on off} | The state you want to set. Specify on or off. |

setsspeakervolume

Environment: C and Java

Syntax: **setsspeakervolume** *apparatusid*/volume=*speakervolume*

Sets the volume for a speaker at the auditory apparatus on the assigned device.

Specify the following parameters:

| | |
|------------------------------|--|
| <i>apparatusid</i> | The identifier for the apparatus for which you want to set the speaker volume. Specify the identifier in hexadecimal. |
| <i>/volume=speakervolume</i> | Specify one of the following for <i>speakervolume</i> : <ul style="list-style-type: none">• minimum• maximum• incremented• decremented Alternatively, specify a value between 1 and 99 for a level of volume that falls between silence and maximum volume. The granularity of the values 1 through 99 and the relationship between volume and loudness are both device specific. |

singlestepconference

Environment: C and Java

Syntax: **singlestepconference** *dn callrefid* {*active*|*silent*}
[*data*]

Immediately joins an additional party to an active call (only for CSTA Phase II switches and the Avaya DEFINITY). At least one of the devices in the active call must be monitored by CTC Test to obtain the reference identifier for the call.

Specify the following parameters:

| | |
|-------------------------|--|
| <i>dn</i> | A DN for one of the devices in the active call |
| <i>callrefid</i> | The reference identifier for the active call in hexadecimal |
| <i>active or silent</i> | <ul style="list-style-type: none">• Specify <i>active</i> for the call to join the conference as a full participant• Specify <i>silent</i> for the call to join the conference, but can only listen |
| <i>data</i> | Optional. A string of up to 127 characters |

singlesteptransfer

Environment: C and Java

Syntax: **singlesteptransfer** *dn* [*callrefid* [*data*]]

Transfers the current call to another number and disconnect the assigned device.

Specify the following parameters:

| | |
|------------------|---|
| <i>dn</i> | The DN of the new device. |
| <i>callrefid</i> | The call reference id of the call. This is automatically included, but you can specify a <i>callrefid</i> (in hexadecimal). |
| <i>data</i> | Optional. A string of up to 127 characters. Note that you must pass a <i>callrefid</i> if you pass a data string. |

snapshot

Environment: C and Java

Syntax: **snapshot**

Gets the status and number of calls at the assigned device.

swapwithheld

Environment: C and Java

Syntax: **swapwithheld** [*heldrefid* [*callrefid*]]

Swaps the current call with a call on consultation hold.

The *heldrefid* and *callrefid* of the calls are automatically included or you can specify the *heldrefid* and *callrefid* (in hexadecimal). If you include a *callrefid*, you must also specify a *heldrefid*.

transfercall

Environment: C and Java

Syntax: **transfercall** [*heldrefid* [*callrefid*]]

Completes the transfer of a call initiated by a *consultationcall* command.

The *heldrefid* and *callrefid* of the calls are automatically included or you can specify the *heldrefid* and *callrefid* (in hexadecimal). If you include a *callrefid*, you must also specify a *heldrefid*.

Writing a CTC Test Script

4.1 Test Script Overview

A CTC Test script is a text file you write containing CTC Test and script control commands. The sequence you choose for these commands defines the tests performed on your call processing application and configuration.

By writing a CTC Test script, you can access features additional to those offered by the individual CTC Test commands, including:

- Iterations
Set the number of times a script or an individual test runs
- Events and timeouts
Direct a test to wait for specific events or timeouts
- Cleanup
Define the cleanup process after an error
- Macros
Define macros that allow text substitution in a script prior to run time
- Logging
Choose the level of, and destination for, logging information

You access these features by use of the appropriate functions offered by the CTC Test scripting facility:

- CTC Test commands
You perform telephony activities by issuing CTC Test commands. Chapter 3 describes the syntax of the CTC Test commands.
- Script control qualifiers
You control the execution of an individual CTC Test command by appending script control qualifiers. Section 4.2 describes the script control qualifiers.

- Script structure commands

You structure the flow of a CTC Test script by including script structure commands. Section 4.3 describes the script structure commands.

- Macro substitution

You can substitute text within a CTC Test script at run time using macros. Section 4.4 describes how to define and use macros.

- Command line qualifiers

You control the overall behavior of a script session by using command line qualifiers when you invoke the script at the CTC Test prompt. Section 2.2 describes the command line qualifiers.

To see how you might combine these functions in a real life case, refer to the example CTC Test scripts in Appendix A.

Follow these guidelines for including commands in your script:

1. Place each command on a separate line.
2. Be careful to observe any case-sensitivity when entering commands in a script. Although the scripting commands themselves are not case-sensitive, some CTC Test commands are (see Chapter 3 for details).
3. Include a clean up procedure if you do not want processing of the next test to begin immediately on receipt of a CTC Test command error.
4. When the script run comes to an end, deassigns are not done automatically. You should specifically include relevant `deassign` commands in your script.

4.2 Script Control Qualifiers

4.2.1 Description

You use script control qualifiers to control the execution of an individual CTC Test command in your script. For example:

```
1 makecall 2300/events=(DEST_SEIZED INBOUND_CALL OFFHOOK)
```

In this case, CTC Test will wait for the three events `DEST_SEIZED`, `INBOUND_CALL`, and `OFFHOOK` after calling the `makecall` command shown.

4.2.2 Functions Available

The script control qualifiers offer these functions, which are described in the rest of this section:

| To do this... | Use this qualifier... |
|---|-----------------------|
| Designate the cleanup routine to run if the CTC Test command returns an error | <code>/clean</code> |
| Pause the test until all the events you specify are returned | <code>/events</code> |
| Stop execution of the script if an error occurs | <code>/stop</code> |
| Set the maximum time to wait for events before returning a timeout error | <code>/timeout</code> |

4.2.3 Definitions

`/clean`

Syntax: `/clean=name`

Designate the cleanup routine to run if the CTC Test command returns an error.

If you do not designate a cleanup routine for a CTC Test command, and an error is returned, CTC Test does one of three things:

1. Runs the cleanup routine you have designated for the whole test.
2. If you have not included a cleanup routine for the test, runs the `cleanup` routine you have included for the whole script.
3. If you have not included a `cleanup` routine for the script, ceases execution of the current test and begins the next.

See Section 4.3 for more details of cleanup procedures, referring in particular to the descriptions of the `clean:` and `test` script structure commands.

`/events`

Syntax: `/events=(eventname[:n] eventname[:n] eventname[:n] ...)`

Pause the test until all the events you specify are returned.

`/events` allows you to synchronize the channels and their call states so that errors do not occur because the switch cannot keep up.

This qualifier can be used with any of the commands described in Chapter 3 and with the `waitevent` command described in Section 4.3.

eventname can be any of the following:

AGENT_LOGGED_OFF
AGENT_LOGGED_ON
AGENT_MODE_CHANGE
BACK_IN_SERVICE
CALL_INFORMATION
DEST_BUSY
DEST_CHANGED
DEST_INVALID
DEST_NOTOBTAINABLE
DEST_SEIZED
DIGITS_COLLECTED (Meridian only)
DIVERTED
END_OF_PLAY (Meridian only)
ERROR
INBOUND_CALL
OFFHOOK
OP_ANSWERED
OP_CONFERENCED
OP_DISCONNECTED
OP_HELD
OP_RETRIEVED
OTHER
OUT_OF_SERVICE
PRIVATE
TP_ANSWERED
TP_CONFERENCED
TP_DISCONNECTED
TP_RETRIEVED
TP_SUSPENDED
TRANSFERRED
UNAVAILABLE

You can specify as many events as you wish.

By default, CTC Test accepts events from any channel. Optionally, you can specify the channel by adding a colon and the channel label *n* to *eventname*. (Note that if you use an asterisk * this also refers to any channel.)

For example, if you wanted the test to pause until the DEST_BUSY event arrived on channel 2, your */events* qualifier would read:

```
/events=(DEST_BUSY:2)
```

Both of the following */events* qualifiers would pause the test until the DEST_BUSY event arrived on any channel:

```
/events=(DEST_BUSY)
```

```
/events=(DEST_BUSY:*)
```

During execution of */events*, error and event information is still displayed on screen. The type of information displayed is as you specified with the */loglevel* qualifier on the CTC Test command line (the default is log level 4). On the display, or in a logfile, an event you specified with the */events* qualifier is marked with an asterisk *.

If the events are not returned within the timeout period, */events* times out and produces an error. (See the */timeout* qualifier for details of timeouts.) CTC Test then executes any cleanup routine you have designated. (See the *clean* command in Section 4.3 for details of cleanup routines.)

/stop

Syntax: */stop*

Stop execution of the script if an error occurs.

/timeout

Syntax: */timeout=n*

Set the maximum time to wait for events before returning a timeout error.

Specify the timeout period in seconds.

This timeout value overrides that set by the */timeout* qualifier on the CTC Test command line.

4.3 Script Structure Commands

4.3.1 Description

You use script structure commands to group CTC Test commands into routines and to control script execution. For example:

```
iterations 10
```

In this case, CTC Test will execute the whole script 10 times.

4.3.2 Functions Available

The script structure commands offer the following functions and are described in the rest of this section:

| To do this... | Use this command or notation... |
|--|---------------------------------|
| Start a comment line | # |
| Continue a CTC Test command on the next line | \ |
| Mark the start and end of a group of CTC Test commands to form a routine | { } |
| Start and name a cleanup routine | clean: |
| Designate the cleanup routine to run if an error occurs during a test | clean= |
| Repeat the following CTC Test commands | count |
| Reverse a <code>noerrors</code> command | errors |
| Execute the entire script a set number of times | iterations |
| Tell CTC Test to ignore any errors | noerrors |
| Mark the start of each test and give it an identifying number | test |
| Wait for the specified event(s) to return before continuing the test | waitevent |
| Insert a fixed delay in the script | waittime |

4.3.3 Definitions

#

Syntax: #

Start a comment line.

Use # at the start of a line to include text that is not part of a test command, for example, any comments you want to attach.

\

Syntax: \

Continue a CTC Test command on the next line.

Enter \ at the end of a line to indicate the command continues on the next line.

{ }

Syntax: { *commands* }

Mark the start and end of a group of CTC Test commands to form a routine.

You use braces { } to mark the beginning and end of *test*, *clean*, and *count* routines. See the descriptions of the *test*, *clean*, and *count* commands for details. Note that each of the braces and each of the commands must be on a line by itself.

clean:

Syntax: *clean:name*

Start and name a cleanup routine.

Use any name you wish for the routine, though note that you should only use the reserved name *cleanup* to label the default cleanup routine for the whole script.

On the line following the *clean* command, mark the start of the routine with an opening brace {. On the line following the last command in the *clean* routine, mark the end of the routine with a closing brace }.

You use cleanup routines to specify the actions taken by CTC Test in response to an error during execution of your script. It is up to you how large a scope you want to give to your cleanup routine. You can make it apply to as little as a single command

or to as much as the whole script, as follows:

- For a single command, use the `/clean` script control qualifier (see Section 4.2).
- For all of a test, use the `clean` command (see the `test` command in this section).
- For a whole script, include a cleanup routine called `cleanup`.

If there is an error, CTC Test calls the cleanup routine with the narrowest scope. In other words, if you specified a cleanup routine for the command it is called rather than a cleanup routine you specified for the whole test. Similarly, the script default `cleanup` routine is only called if you did not specify a cleanup routine for either the command or the test.

If you do not specify any applicable cleanup routine, the current test will stop on an error and CTC Test will take no other actions before running the next test.

clean=

Syntax: `clean=name`

Designate the cleanup routine to run if an error occurs during a test.

You must include the `clean=` command within the test routine.

See the description of the `test` command for more details on the use of test routines. See the description of the `clean:` command for more details on the use of cleanup routines.

count

Syntax: `count:n`

Repeat the following CTC Test commands.

Indicate the number of times to repeat in *n*.

You can nest `count` routines within other `count` routines and within `test` routines. This allows you to repeat groups of commands within a single test.

On the line following the `count` command, mark the start of the routine with an opening brace `{`. On the line following the last command in the `count` routine, mark the end of the routine with a closing brace `}`.

errors

Syntax: `errors`

Reverse a `noerrors` command.

iterations

Syntax: `iterations n`

Execute the entire script a set number of times.

If this command is not present then the script runs only once. Note that the `/iterations` qualifier on the CTC Test command line overrides this command.

noerrors

Syntax: `noerrors`

Tell CTC Test to ignore any errors.

Note that CTC Test will still respond to an event timeout and will continue to log errors.

Use the `noerrors` command at the start of a cleanup procedure so that it executes without interruption. Use the `errors` command at the end of the cleanup procedure to reverse the effect.

test

Syntax: `test n`

Mark the start of each test and give it an identifying number.

Enter this command at the start of each test within the script file. Each test is made up of one or more test commands.

On the line following the `test` command, mark the start of the routine with an opening brace `{`. On the line following the last command in the `test` routine, mark the end of the routine with a closing brace `}`.

If you want to designate a cleanup routine applicable to the whole test, include the following line:

```
clean=name
```

where *name* is the name of your cleanup routine. See the description of the `clean:` command for more details on the use of cleanup routines.

waitevent

Syntax: `waitevent/events=(eventname[:n] eventname[:n] ...)`

Pause the test until all the events you specify are returned.

This command and qualifier provide the same function as the `/events` qualifier you use with standard CTC Test commands (described in Section 4.2). Use it when you

do not want to call a telephony command (such as `makecall`) but you still want to pause your test until all the events you specify are returned.

`waitevent/events` allows you to synchronize the channels and their call states so that errors do not occur because the switch cannot keep up.

For a description of the `/events` qualifier, *eventname* values, and how to clean up events, refer to the description of the `/events` qualifier in Section 4.2.

waittime

Syntax: `waittime n`

Insert a fixed delay in the script.

Enter the delay you require in seconds.

Use the `waittime` command to cause CTC Test to cease execution for an explicit length of time. You can use this as a mechanism to give the system time to return to a settled state after processing commands. For example, you might use `waittime` after issuing a `respondtoroutequery` command, which inherently has to wait for other events to occur before it can complete.

4.4 Macro Substitution

Macro substitution allows you to replace text in your CTC Test script at run time. You can use this function to make your script more flexible. For example, you can write a general purpose test script into which you substitute specific parameters as required.

To implement a macro substitution, follow these steps, which are described in more detail in the rest of this section:

1. Declare the macro within the CTC Test script
2. Define the substitute text in a macros file (see subsection, **Macros File**)
3. Invoke the macros file when you run your CTC Test script

Note that your definition of the substitute text in the macros file can itself contain macros, as described in Section 4.4.2, **Define**.

Macros File

A macros file consists of one or more similar definitions, each on a separate line. You can also include comments by starting the line with a `#`.

By having more than one macros file, you can define different substitute text for the same label. You invoke the specific macros file you require on the CTC Test command

line.

4.4.1 Declare

To declare a macro within the CTC Test script, use this form:

```
$(label)
```

where *label* is the name you attach to the substitute text when you define it in your macros file.

Enter the declaration at each point in the CTC Test script where you want the text substituted.

4.4.2 Define

To define the substitute text in the macros file, use this form:

```
label=substitute_text
```

where *label* is the name you use to declare the macro in your CTC Test script and *substitute_text* is the text you want substituted.

You can also use macros within the *substitute_text* definition (that is, the text can be a macro itself). For example, here is a macro declaration in a CTC Test script:

```
$(command1)
```

In the macros file, the substitute text is defined as follows:

```
command1=makecall $(arguments1)  
arguments1=2300
```

After parsing, the CTC Test command runs as:

```
makecall 2300
```

4.4.3 Invoke

To invoke the macros file you require, use the `/macros` qualifier on the CTC Test command line (see Section 2.2 for details). During parsing, CTC Test replaces the macro declaration in the CTC Test script with the substitute text you defined in the macros file.

CTC Test issues a warning if a macro declared in the CTC Test script is not defined in the macros file. CTC Test replaces the macro with an empty string. Parsing then continues, unless you specified the `/stop` qualifier on the CTC Test command line.

After parsing, CTC Test executes the CTC Test script.

4.4.4 Example Use of a Macro

Here is an example of the use of a macro:

1. Declare the `dn` macro in the CTC Test script:

```
1 makecall $(dn)
```

2. Define the `dn` macro in a macros file:

```
dn=2300
```

3. After parsing, the CTC Test script command runs as:

```
1 makecall 2300
```

For more detailed examples, refer to the sample CTC Test scripts in Appendix A.

Example CTC Test Scripts

A.1 Example CTC Test Script

```
#
# This is an example of a script file for ctc_test.
#

iterations 10

clean:cleanup
{
noerrors
1 deassign
2 deassign
errors
}

clean:makecall
{
noerrors
1 deassign
2 deassign
errors
}

# Make a call and hangup

Test 1
{
    1 assign $(Dn1) $(server) $(LogId) $(Transport)
    1 setmon on
    1 getevent

    2 assign $(Dn2) $(Server) $(LogId) $(Transport)
```

```

2 setmon on
2 getevent

Count:10
{
    1 makecall $(Dn2)/events=$(InboundEvents) \
                /timeout=5 \
                /clean=makecall

    2 answer /events=(TP_ANSWERED OP_ANSWERED) \
            /timeout=3 \
            /clean=makecall

    1 hangup /events=(TP_DISCONNECTED OP_DISCONNECTED) \
            /timeout=3 \
            /clean=makecall

    waittime 1
}
1 deassign
2 deassign
}

```

A.2 Example CTC Test Macro File

```

#
# Example macro file to customize the example script file.
#

Dn1=2001
Dn2=2002
Server=elbrig
LogId=simulator
Transport=ncacn_ip_tcp

InboundEvents=DEST_SEIZED INBOUND_CALL OFFHOOK

```