
CCL Modem Scripting Guide

Drivers, Kernel, & Hardware: User-Space Device Access



2007-06-28



Apple Inc.
© 2007 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

.Mac is a registered service mark of Apple Inc.

Apple, the Apple logo, Cocoa, eMac, Mac, Mac OS, and Macintosh are trademarks of Apple Inc., registered in the United States and other countries.

DEC is a trademark of Digital Equipment Corporation.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE

ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Introduction **Introduction to CCL Modem Scripting Guide 7**

- Organization of This Document 7
- What You Need to Get Started 7
- Available Modem Scripts 8

Chapter 1 **Writing a CCL Script 9**

- Initiating a Call 10
- Answering a Call 12
- Terminating a Call 13

Chapter 2 **CCL Script Syntax 15**

- CCL Commands 15
- Comments 15
- Capitalization 16
- Labels 16
- String Formats 16
- Variable Strings (varStrings) 17
- Match Strings 18
- Script Size 19

Chapter 3 **CCL Bundles and Property Lists 21**

- About mlts Resource Storage 21
- About Property List Storage 21
- Property List Structure 22
 - CCL Personalities Dictionary 22
 - Miscellaneous Top-Level Property Keys 23
- Example Property Lists 24

Appendix A **CCL Command Reference 27**

- Comments 27
- Labels and Sections 27
 - @ANSWER 27
 - @CCLSCRIPT 27
 - @HANGUP 28
 - @LABEL 28
 - @ORIGINATE 28
- CCL Commands 29

ASK 29
CHRDELAY 29
COMMUNICATINGAT 30
DETRIES 30
DTRCLEAR 30
DTRSET 30
EXIT 31
FLUSH 31
HSRESET 31
IFANSWER 32
IFORIGINATE 32
IFSTR 33
IFTRIES 33
INCTRIES 34
JSR 34
JUMP 34
LBREAK 35
MATCHCLR 35
MATCHREAD 35
MATCHSTR 35
MONITORLINE 36
NOTE 36
PAUSE 37
RETURN 38
SBREAK 38
SERRESET 38
SETSPEED 38
SETTRIES 39
USERHOOK 39
WRITE 40

Appendix B Result Codes 41

Appendix C Cable Specifications 43

Modem Control Issues 43
Recommended Modem Control 44

Document Revision History 45

Tables and Listings

Chapter 3 **CCL Bundles and Property Lists** 21

- Listing 3-1 A GPRS device property list example 24
- Listing 3-2 A dialup modem property list example 25

Appendix C **Cable Specifications** 43

- Table C-1 Computer to Modem Cable Specifications 43

Introduction to CCL Modem Scripting Guide

Mac OS X supports communication over telephone lines using Hayes-compatible modems and similar communication channels such as cellular phones. It provides support for these modems using modem scripts.

Mac OS X comes with a number of modem scripts preinstalled. To use Mac OS X with a modem or cellular phone for which a script is not supplied, the user or the modem vendor must either obtain or write a script to control the modem.

Modem scripts are written using the Communication Command Language (CCL). You can create these scripts programmatically using the iSync Plug-in Maker Tool (described in the *iSync Plug-in Maker User Guide*) or manually using any text editor.

This guide includes instructions for writing scripts and descriptions of all the CCL commands. It is intended for experienced programmers with a good understanding of telecommunications and modem operation.

Organization of This Document

The guide is divided into two chapters and two appendixes:

- [“Writing a CCL Script”](#) (page 9) describes the basic elements and structure of a CCL file and the basic tasks a script must perform.
- [“CCL Command Reference”](#) (page 27) lists the CCL commands, providing for each a definition, syntax, and an example, if appropriate.
- [“Result Codes”](#) (page 41) lists result codes returned by the CCL, with a description of the error and the accompanying message, if any.
- [“Cable Specifications”](#) (page 43) discusses requirements for a CTS/RTS handshaking cable, desirable when using Mac OS X with a 9600 bps or faster modem.

What You Need to Get Started

To write a CCL script for Mac OS X, your computer should be running Mac OS X v10.5. For ease of writing scripts, you should also have the latest version of the Mac OS X Developer Tools installed. These provide the iSync Plug-in Maker tool, which makes it easier to create and edit scripts.

Important: You will also need the documentation that came with your modem; many commands vary from one modem to another.

For information about the iSync Plug-in Maker, see the *iSync Plug-in Maker User Guide*.

You may also find it useful to consult a reference manual on telecommunications and modems, such as *The Complete Modem Reference* by Gilbert Held, published by John Wiley & Sons, Inc.

Available Modem Scripts

A number of modem scripts have already been written for use with Mac OS X. These can be found in the Modem Scripts folder within the user, system, and global Library folders (`/Library/Modem Scripts`, for example). If you have one of the modems for which a script has been provided, you don't need to write a script. You can display a list of the provided scripts from within the iSync Plug-in Maker tool, as described in the *iSync Plug-in Maker User Guide*.

If you need to write a script, you may be able to use an existing script as a template. Be sure to use the Save As command to make a copy of the script you're modifying, so that you don't overwrite the original script.

Writing a CCL Script

This chapter describes the basic tasks a CCL script must perform. For detailed descriptions of the modem scripting commands, see “[CCL Command Reference](#)” (page 27).

A modem script executes in one of two possible modes, each with a separate entry point. The modes are as follows:

originate

used when a call is initiated.

answer

used when call answering is enabled. Mac OS X does not provide a user interface for enabling or using answer mode.

hangup

used to terminate every connection, whether in originate or answer mode, and whether or not the connection was successful.

The following figure provides an overview of the tasks your script must perform in each mode. The remainder of this chapter describes these tasks.

Originate mode (to initiate a call) @ORIGINATE entry point	Answer mode (to answer a call) @ANSWER entry point	Hangup mode (to terminate a call) @HANGUP entry point
Configure serial port.	Configure serial port.	Turn off CTS hardware handshaking.
Turn off CTS hardware handshaking.	Turn off CTS hardware handshaking.	Issue hangup command.
Configure modem.	Configure modem.	If unsuccessful, put modem in command mode and issue hangup again.
Dial.	Enable auto-answering.	Recall factory defaults.
Wait for response.	Detect ring.	Turn off auto-answering.
If call fails, exit with error result code.	If call fails, exit with error result code.	Exit.
If call succeeds, display message.	If call succeeds, display message.	
If modem error correction established, issue <code>USERHOOK 2</code> or <code>USERHOOK 4</code> .	If modem error correction established, issue <code>USERHOOK 2</code> or <code>USERHOOK 4</code> .	
If modem data compression established, issue <code>USERHOOK 3</code> .	If modem data compression established, issue <code>USERHOOK 3</code> .	
Configure serial port.	Configure serial port.	

Originate mode (to initiate a call) @ORIGINATE entry point	Answer mode (to answer a call) @ANSWER entry point	Hangup mode (to terminate a call) @HANGUP entry point
If appropriate, turn on CTS hardware handshaking.	If appropriate, turn on CTS hardware handshaking.	
Issue <code>USERHOOK 1</code> .	Issue <code>USERHOOK 1</code> .	

Initiating a Call

When Mac OS X initiates a call, it executes the script starting at the `@ORIGINATE` entry point. The script must perform the following tasks to initiate a call:

1. Configure the communication channel.

Use the `SERRESET` command to reset the communication parameters of the serial port (or emulated equivalent communication channel). Set the speed of the connection to the maximum speed of the device. (You may change the serial port speed later in the script.) Set the number of bits to be used for stop, start, and parity.

Use the `HSRESET` command to turn off the serial port's flow control options. (You will turn on the appropriate options later in the script.)

2. Configure the device.

Following is a list of standard steps for configuring a modem-like communication device. Check the documentation that came with your device to determine whether it requires different steps.

- a. Recall the factory default configuration settings.
- b. If your device has a speed of 14,400 bps or higher, you need to configure the device for RTS/CTS handshaking. For modems, an appropriate cable must be used, as described in [“Cable Specifications”](#) (page 43). Do not turn on hardware handshaking until the connection has been made.
- c. Configure the modem for DTR usage.
- d. Turn local echo off. When local echo is on, the modem sends commands it receives back to the computer.
- e. Set the modem to return detailed result codes including the speed of the connection and the results of error correction and data compression negotiation.
- f. If the modem can do error correction, set error correction according to `varString4`.

If `varString4` is set to 0, turn error correction off.

If `varString4` is set to 1, have the modem negotiate the best available error correction with the remote modem. If no error correction can be established, have the modems remain connected without error correction.

If `varString4` is set to 2, have the modem try to establish MNP10 error correction with the remote modem. If MNP10 negotiation fails, have the modems remain connected without error correction.

Note: If the modem uses a proprietary error correction protocol, make sure that it will try to negotiate standard protocols if it is unable to negotiate the proprietary protocol. If not, disable error correction. Do not disable the Trellis error protocol; it is part of the V.32 standard.

- g. Mac OS X is generally more efficient than a modem or similar device at compressing data. If you believe you have a special situation in which hardware data compression is preferable, have the script set up the device to negotiate data compression.
- h. Turn the speaker on or off according to the value of `varString2`.

3. Dial the phone number.

- a. Have the script check whether the dial string extends into `varString8` (then `varString9`) by using the `IFSTR` command to check for a blank string. If the entire dial string fits in `varString7`, have the script issue a single dial command. If the dial string is longer than `varString7`, have the script issue multiple dial commands referencing `varString7`, `varString8`, and if necessary, `varString9`.

- b. Set tone or pulse dialing in the dial command according to the value of `varString3`.

- c. If `varString6` is set to 1, have the device begin dialing without confirmation of dial tone detection. This is useful when the phone system provides a nonstandard dial tone that can't be recognized by a modem's tone detection circuitry.

If `varString6` is set to 2, the user has already dialed the remote number. Have the script cause the modem to retrain with the remote system. This is useful when the dialing sequence is too complex or interactive for the CCL script to navigate.

- d. Display the dialed phone number in the Internet Connect status window and the activity log using the `NOTE` command. Use `varString1` in log messages rather than a concatenation of `varString7`, `varString8`, and `varString9`.

4. Wait for the device to respond.

5. If the call fails, return an error result code indicating what happened.

For example, use error result code -6022 if the line is busy. (See "Result Codes" (page 41) for a complete list.)

6. If the call is successful, indicate that a connection has been established.

Display a message such as "Communicating at 9600 bps" in the Internet Connect status window using the `NOTE` command. Also display messages indicating the results of error correction and data compression negotiation, if applicable.

If you were successful in establishing error correction, issue `USERHOOK 4` (for MNP10 error correction) or `USERHOOK 2` (for all other error correction) to advise Mac OS X that a reliable link was established.

If you were successful in establishing data compression, issue `USERHOOK 3` to advise Mac OS X to turn off its built-in data compression.

7. Configure the communication channel based on the device speed and the connection rate.

If your device has a speed of 14,400 bps or higher (and, for modems, if you are using a RTS/CTS handshaking cable as described in “[Cable Specifications](#)” (page 43)), use the `HSRESET` command to set flow control to `outputCTS`. Use the `COMMUNICATINGAT` command to tell Mac OS X the connection speed so that it can set its timers appropriately.

If your device has a speed of 9600 bps or slower, use the `SERRESET` command to reset the serial port speed to the speed of the connection.

8. Exit the script so that Mac OS X can use the connection.

For more details on the commands and variables described in this section, see “[CCL Command Reference](#)” (page 27).

Answering a Call

To answer a call, Mac OS X executes the script starting at the `@ANSWER` entry point. The script must perform the following tasks to answer a call:

1. Configure the communication channel.

Use the `SERRESET` command to reset the communication parameters of the serial port (or emulated equivalent communication channel). Set the speed of the connection to the maximum speed of the device. (You may change the serial port speed later in the script.) Set the number of bits to be used for stop, start, and parity.

Use the `HSRESET` command to turn off the serial port's flow control options. (You will turn on the appropriate options later in the script.)

2. Configure the device.

It will require several commands to completely configure the device. To prevent calls being answered before the configuration is complete, disable auto-answering in the first command the script issues. (You will enable it in step 3.)

For additional details about configuring the device, see step 2 of “[Initiating a Call](#)” (page 10) earlier in this chapter.

- a. Recall the factory default configuration settings.
- b. If your device has a speed of 14,400 bps or higher, set up the device for RTS/CTS handshaking. For modems, use an appropriate cable, as described in “[Cable Specifications](#)” (page 43).
- c. Configure the device for DTR usage.
- d. Turn local echo off.
- e. Set the device to return detailed result codes including the speed of the connection and the results of error correction and data compression negotiation.
- f. If the device can do error correction, set error correction according to `varString4`.

- g. Mac OS X is generally more efficient than a modem at compressing data. If you believe you have a special situation in which hardware data compression is preferable, have the script set up the modem to negotiate data compression.
- h. Turn the speaker on or off according to the value of `varString2`.

3. Enable auto-answering and wait for the result.

On an incoming call, the device issues a `RING` result code.

4. If the call fails, return an error result code indicating what happened.

For example, use error result code `-6021` if the device cannot detect a carrier signal on the phone line. (See [“Result Codes”](#) (page 41) for a complete list.)

5. If the call is successful, indicate that a connection has been established.

Display a message such as “Communicating at 9600 bps” in the Internet Connect status window using the `NOTE` command. Also display messages indicating the results of error correction and data compression negotiation, if applicable.

If you were successful in establishing hardware error correction, issue `USERHOOK 4` (for MNP10 error correction) or `USERHOOK 2` (for all other error correction) to advise Mac OS X that a reliable link was established.

If you were successful in establishing data compression, issue `USERHOOK 3` to advise Mac OS X to turn off its built-in data compression.

6. Configure the communication channel based on the device speed and the connection rate.

Issue the `USERHOOK 1` command. The `USERHOOK 1` command informs Mac OS X that the serial port or equivalent is busy answering a call, which prevents Mac OS X from giving it up to another application.

If your device has a speed of 14,400 bps or higher (and, for modems, if you are using a RTS/CTS handshaking cable as described in [“Cable Specifications”](#) (page 43)), use the `HSRESET` command to set flow control for `outputCTS`.

Use the `COMMUNICATINGAT` command to tell Mac OS X the connection speed so that it can set its timers appropriately.

If your device has a speed of 9600 bps or slower, use the `SERRESET` command to reset the serial port speed to the speed of the connection.

7. Exit the script so that Mac OS X can use the connection.

For more details on the commands and variables described in this section, see [“CCL Command Reference”](#) (page 27).

Terminating a Call

To terminate a call, Mac OS X executes the script starting at the `@HANGUP` entry point.

The hangup part of the script is executed to terminate a connection whenever the @ORIGINATE or @ANSWER parts of the script have been executed, regardless of the result. The hangup part of the script is also executed when Mac OS X terminates answer mode.

The script must perform the following tasks to terminate a call:

1. **If hardware handshaking is used in the @ORIGINATE or @ANSWER part of the script, turn off hardware handshaking.**

Use the HSRESET command to turn off hardware handshaking.

2. **If possible, cause the device to enter command mode.**

Before you issue a hangup command, you may need to get the attention of the device by, for example, issuing a short break, a long break, or the attention sequence "+++", or by toggling DTR. Consult your device documentation for the appropriate method.

3. **Issue a hangup command.**

4. **Recall the factory default configuration settings.**

Since you recalled the default settings at the beginning of the script, this is not necessary if the only communications application you use is Mac OS X's built-in modem support; however, recalling the default settings at the end of your script is recommended in case the next communications application that you use does not take care of this itself.

5. **Turn off auto-answering.**

This prevents the device from answering the phone until call answering is enabled.

6. **Exit with an appropriate message.**

If successful, return a result code of 0. If unsuccessful, return the appropriate error result code as listed in "Result Codes" (page 41)

For more details on the commands and variables described in this section, see "CCL Command Reference" (page 27).

CCL Script Syntax

A modem script is a set of instructions that tells a computer how to interact with a modem so that calls can be initiated and received. To establish a connection, a script typically configures and then connects the modem. To terminate a connection, the script disconnects the modem by hanging up and then restores the modem settings that were in effect before the call.

Each type of modem used with Mac OS X requires a modem script. Many scripts are provided with Mac OS X. (See [“Available Modem Scripts”](#) (page 8) in the Introduction of this document for more information.)

If no script is provided for your modem, you must write one using Communication Command Language (CCL), a programming language that configures and controls your modem. This section describes the basic elements and structure of a CCL file.

CCL Commands

Each line of CCL code consists of one instruction that is made up of a command and its parameters, if any. Modem commands are used as parameters of CCL commands. For example, the command `write "ATDT^1\13"` includes the following:

- a CCL command, `write`
- a modem command, `ATDT`
- a modem command parameter, `^1\13`

This command tells the CCL interpreter to send to the modem the modem command `ATDT` followed by variable string `#1`, and a carriage return (ASCII code 13).

The CCL interpreter reads scripts from left to right and from top to bottom. It reads straight through, from beginning to end, unless you tell it otherwise (for example, by using the `JUMP` command).

For a complete list of commands and their usage, see [“CCL Command Reference”](#) (page 27).

Comments

You can insert explanatory comments into your script. To enter a comment, start the line with an exclamation point (!).

You may also want to use a blank comment line to make your script more readable; to do so, type an exclamation point with no text and press Return.

Capitalization

The CCL interpreter is not case sensitive. Therefore, `@LABEL 1`, `@Label 1`, `@label 1`, and `@laBe1 1` are all valid (and equivalent) instructions.

Labels

Labels are used to mark locations in the script. Other script commands, such as `JUMP`, transfer control to locations in the script marked by the `@LABEL` command. For instance, `JUMP 13` tells the CCL interpreter to jump to label 13 and start executing the commands after the `@LABEL 13` command. A script may use up to 128 labels, numbered 1–128.

String Formats

To delimit a string in CCL code, you can use single quotation marks (') or double quotation marks ("). If you do not start the string with a single or double quotation mark, any of the following characters determines the end of the string: space, return, tab, comma, or semicolon.

CCL strings may include references to variable strings. See [“Variable Strings \(varStrings\)”](#) (page 17) later in this chapter for details.

CCL strings may include nonprinting characters such as null, tab, and return. To support these nonprinting characters, the CCL interpreter recognizes two special characters: the backslash (\) and the caret (^).

The backslash is a quote character. You can use the backslash to include a nonprinting character by specifying the decimal representation of the ASCII character (decimal numbers 00 to 99 are valid) or to explicitly include the backslash or caret character in a string.

The caret is a variable delimiter character. You can use it to insert a variable string or an ASK string into another string. (For details, see [“Variable Strings \(varStrings\)”](#) (page 17) later in this chapter.)

Here are some examples of how the backslash and caret are used:

String	Result
<code>\13</code>	inserts a carriage return (0x0D) into the string
<code>\00</code>	inserts the null character (0x00) into the string
<code>\\</code>	inserts the backslash (\) character (0x5C) into the string
<code>\^</code>	inserts the caret (^) character (0x5E) into the string
<code>^1</code>	inserts variable string 1 into the string
<code>^*</code>	inserts the ASK string into the string

Here are some string examples:

String contents	Printed Output
'this is a test'	this is a test
thisisatest	thisisatest
"this is a test"	this is a test
this is a test	this
"this \34\73\83\34 a test"	this "IS" a test
"ATDT^1"	ATDT555-1212(if variable string 1 is 555-1212)

Variable Strings (varStrings)

CCL strings may include references to variable strings, which are strings passed to the script as parameters. Most of these values are provided by Mac OS X based on user-specified parameters.

Mac OS X uses the following variable strings:

Variable String	Contents	Values
varString1	The complete dial string (the telephone number to dial, with the necessary prefixes and suffixes).	
varString2	The modem speaker flag:	0—Speaker off. 1—Speaker on.
varString3	The tone/pulse dialing selector:	T—Tone dialing. P—Pulse dialing.
varString4	The modem error correction flag:	0—Script should not try to use modem error correction. 1—Script should try to use modem error correction. 2—Script should try to establish MNP10 error correction.
varString5	Reserved for future use.	

Variable String	Contents	Values
<code>varString6</code>	The dialing mode flag:	<p>0—Normal dialing.</p> <p>1—Blind dialing (that is, have the modem begin dialing without confirmation of dial tone detection).</p> <p>2—Manual dialing (that is, have the script assume that the user has already picked up the phone and dialed the remote number).</p>
<code>varString7–varString9</code>	Variable strings 7–9 break a long dial string into shorter strings for use with modems that can accept only a limited command string size. They are automatically generated by Mac OS X.	<p>The variables <code>varString7</code>, <code>varString8</code>, and <code>varString9</code> contain the contents of <code>varString1</code> divided into shorter pieces. Each of these strings is limited to 40 characters in length. Additional characters that will not fit into <code>varString7</code> spill over automatically into <code>varString8</code>, and so on.</p> <p>If one or more of these three variables are unused, the empty variables contain a blank string (ASCII 20 hex). See “About mlts Resource Storage” (page 21) for legacy information about these strings.</p>
<code>varString27–varString30</code>	Defined in the information property list in your CCL script bundle. See “CCL Bundles and Property Lists” (page 21) for information.	

All eight variable strings are passed to the script when it is running in originate mode. The modem speaker flag (`varString2`) and the error correction flag (`varString4`) are passed to the script when it is running in answer mode.

Match Strings

The CCL interpreter has a buffer that can hold up to 32 strings loaded by the `MATCHSTR` command. The `MATCHCLR` command erases the contents of the buffer. The `MATCHREAD` command reads input from the modem or other communication device and compares the input to the strings currently in the buffer. If a match is found in the specified time, execution continues at the label associated with that match string.

A recommended strategy for sending commands to the modem is as follows:

- Use `MATCHCLR` to clear all match strings.
- Use `MATCHSTR` to load match strings with appropriate responses.
- Use `WRITE` to send commands to the modem.

- Use `MATCHREAD` to check for defined modem responses. If an appropriate response is received, branch as defined by the corresponding `MATCHSTR` command.
- Use `SETTRIES`, `INCTRIES`, and `IFTRIES` to loop a few times.
- If an appropriate response is not received, branch to exit or to alternate code as defined by the `MATCHREAD` command.

Script Size

Scripts may be a maximum of 32000 bytes. This is large enough for relatively complex dial scripts. However, if your script is too large, you may be able to make it small enough by minimizing the number and length of comments.

CCL Bundles and Property Lists

Mac OS X v.10.5 and later supports storage of strings and other data outside of the modem script itself in an easily modified property list file. Classic Mac OS (System 7 through Mac OS 9) and Mac OS X prior to v.10.5 provided similar support with data stored in the resource fork of each script. This appendix explains both mechanisms for external storage to allow ease of conversion into the more modern property list format.

About mlts Resource Storage

A modem script from Mac OS 9 and earlier may contain an optional `mlts` resource that tells the CCL interpreter the characteristics of the modem. Byte 1 indicated support for at least one type of *standard* error correction (1=supported). Byte 2 was reserved. Byte 3 indicated contained the maximum length (in characters) for variable string 7, byte 4 for variable string 8, and byte 5 for variable string 9.

These legacy `mlts` options are not supported in Mac OS X. Mac OS X assumes that all modems can support dial strings of at least 40 characters per dial string and that all modems support built-in error correction.

About Property List Storage

Beginning in Mac OS X v. 10.5, the preferred format for external storage is as a property list. When using this mechanism, the CCL script must be enclosed in a package. The structure of such a package is shown below:

```
MyModem.ccl
|
|-Contents
|   |
|   |- Info.plist      <- The property list
|   |- Resources
|   |   |-MyModem     <- The script itself
```

For general information about property lists, see *Property List Programming Topics for Core Foundation* and *Property List Programming Guide*. The details of what you should put in the property list are covered in the following sections.

Property List Structure

CCL Personalities Dictionary

CCLParameters Dictionary

This dictionary (`dict` element) contains parameters that are made available to your CCL script through variable strings. (See “[Variable Strings \(varStrings\)](#)” (page 17) for more information on how to use these variables in your script.)

Key in CCLParameters Dictionary	Variable String in CCL Script	Description
Connect Speed	^20	The serial port speed desired. You may use -1 if your device has no notion of multiple connection speeds.
Init String	^21	The device initialization string.
Preferred APN	^22	Preferred access point name. Used mainly for cellular phone connections.
Preferred CID	^23	Preferred connection identifier. Used mainly for cellular phone connections.
varString 27	^27	One of four device-specific strings that you can pass in. By using these variable strings, you can share a single script across multiple similar modems or for using the same modem-like device with (for example) multiple cellular services.
varString 28	^28	One of four device-specific strings that you can pass in. By using these variable strings, you can share a single script across multiple similar modems or for using the same modem-like device with (for example) multiple cellular services.
varString 29	^29	One of four device-specific strings that you can pass in. By using these variable strings, you can share a single script across multiple similar modems or for using the same modem-like device with (for example) multiple cellular services.
varString 30	^30	One of four device-specific strings that you can pass in. By using these variable strings, you can share a single script across multiple similar modems or for using the same modem-like device with (for example) multiple cellular services.

Connect Type Key

This `string` element describes the connection type. Valid values are `Dialup` and `GPRS`.

Device Names Dictionary

This `dict` element should contain the following key/value pairs:

- `DeviceModel`—A `string` element representing the device model number or name.
- `DeviceVendor`—A `string` element describing the vendor name.

These strings are presented to the user when choosing a device model and manufacturer.

GPRS Capabilities Dictionary

This `dict` element supports the following keys:

- `CID Query`—A boolean value that tells whether a device allows Mac OS X to query it for valid connection IDs. Mac OS X uses this information outside the scope of your script.
Legal values are `<true/>` and `<false/>`.
- `Data Mode`—A boolean value that indicates whether a device supports initiating a connection by requesting data mode (`AT+CGDATA`). Your script should respect this value when making a connection and use data mode only on communication hardware that supports it.
Legal values are `<true/>` and `<false/>`.
- `Dial Mode`—A boolean value that indicates whether a device supports initiating a connection by dialing a fictitious number (`ATD *99`). If true, Mac OS X will fill in the phone number automatically.
Legal values are `<true/>` and `<false/>`.
- `Independent CIDs`—A boolean value that indicates whether or not a device supports independent CIDs for computer use versus internal (cellular phone feature) use.
When you issue an AT command that sets the CID (`AT+CGDCONT=...`), devices with independent CID support treat the AT command as a temporary change from the default value. Devices that lack this feature "forget" their previous CID value and must be reconfigured upon disconnect to avoid service loss, diminished service, or unexpected billing charges.
Legal values are `<true/>` and `<false/>`.
- `Maximum CID`—A value of type `integer` that represents the maximum allowable connection identifier (CID) value supported by the device. Mac OS X uses this information outside the scope of your script.

Script Name Key

This `string` element tells the name of the script within the `Resources` directory inside your script bundle.

Miscellaneous Top-Level Property Keys

CCL Version Key

This `integer` key should currently always be 1.

CFBundle Keys

The following CFBundle keys should be provided:

- CFBundleDevelopmentRegion
- CFBundleGetInfoString
- CFBundleIdentifier
- CFBundleInfoDictionaryVersion
- CFBundleName
- CFBundlePackageType
- CFBundleShortVersionString
- CFBundleSignature
- CFBundleVersion

For more information about these keys, see *CFBundle Reference*.

Example Property Lists

This section shows two example property lists. Listing 3-1 shows an example of a property list for a GPRS device. Listing 3-2 (page 25) shows a property list for a traditional dialup modem device.

Listing 3-1 A GPRS device property list example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>CCL Personalities</key>
  <dict>
    <key>Default Personality</key>
    <dict>
      <key>Device Names</key>
      <array>
        <dict>
          <key>DeviceModel</key>
          <string>GPRS Device</string>
          <key>DeviceVendor</key>
          <string>Example</string>
        </dict>
      </array>
      <key>Connect Type</key>
      <string>GPRS</string>
      <key>Script Name</key>
      <string>GPRS Example</string>
      <key>GPRS Capabilities</key>
      <dict>
        <key>Data Mode</key>
```

```

    <false/>
    <key>Dial Mode</key>
    <true/>
    <key>Independent CIDs</key>
    <false/>
    <key>CID Query</key>
    <false/>
    <key>Maximum CID</key>
    <integer>10</integer>
  </dict>

  <!-- SystemConfiguration values can replace CCLParameters -->
  <key>CCLParameters</key>
  <dict>
    <key>Init String</key>
    <string>E0V1&amp;F&amp;D2&amp;C1&amp;C2S0=0</string>
    <key>Preferred APN</key>
    <string>network</string>
    <key>varString 27</key>
    <string></string>
    <key>varString 28</key>
    <string></string>
    <key>varString 29</key>
    <string></string>
    <key>varString 30</key>
    <string></string>
  </dict>
</dict>
<key>CFBundleIdentifier</key>
<string>com.apple.ccl.GPRS_Example</string>
<key>CFBundleName</key>
<string>GPRS Example</string>
<key>CCL Version</key>
<integer>1</integer>
<key>CFBundleDevelopmentRegion</key>
<string>English</string>
<key>CFBundlePackageType</key>
<string>CCLB</string>
<key>CFBundleShortVersionString</key>
<string>1.0</string>
<key>CFBundleSignature</key>
<string>iSPM?</string>
<key>CFBundleVersion</key>
<string>1.0</string>
</dict>
</plist>

```

Listing 3-2 A dialup modem property list example

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>CCL Personalities</key>
  <dict>

```

```

<key>Default Personality</key>
<dict>
  <key>Device Names</key>
  <array>
    <dict>
      <key>DeviceModel</key>
      <string>Dialup Device</string>
      <key>DeviceVendor</key>
      <string>Example</string>
    </dict>
  </array>
  <key>Connect Type</key>
  <string>Dialup</string>
  <key>Script Name</key>
  <string>Dialup Example</string>
  <key>CCLParameters</key>
  <dict>
    <key>Connect Speed</key>
    <string>115200</string>
    <key>Init String</key>
    <string>&#xA0;FE0V1</string>
    <key>varString 27</key>
    <string></string>
    <key>varString 28</key>
    <string></string>
    <key>varString 29</key>
    <string></string>
    <key>varString 30</key>
    <string></string>
  </dict>
</dict>
</dict>

<key>CFBundleIdentifier</key>
<string>com.apple.ccl.Dialup_Example</string>
<key>CFBundleName</key>
<string>Dialup Example</string>
<key>CCL Version</key>
<integer>1</integer>
<key>CFBundleDevelopmentRegion</key>
<string>English</string>
<key>CFBundlePackageType</key>
<string>CCLB</string>
<key>CFBundleShortVersionString</key>
<string>1.0</string>
<key>CFBundleSignature</key>
<string>iSPM?</string>
<key>CFBundleVersion</key>
<string>1.0</string>
</dict>
</plist>

```

CCL Command Reference

This chapter describes the CCL commands interpreted by Mac OS X. The commands are presented in alphabetical order. Each command section contains a description of the command; the syntax of the command, including any parameters; and an example, if appropriate.

Note: While the CCL scripting language supported in Mac OS X v. 10.5 is substantially similar to the CCL scripting language supported in Macintosh System 7 (using Apple Remote Access) through Mac OS X v.10.4, there are significant differences in how initialization strings and other external data are stored.

For ARA scripts (prior to Mac OS X), this information was stored in the resource fork of the script itself. In Mac OS X v.10.4 and earlier, this information must be hard-coded into the script in the appropriate places. In Mac OS X 10.5 and later, the information may be optionally placed in a property list (plist). These differences are noted where appropriate.

Comments

To insert a comment or a blank line in the script, start the line with an exclamation point.

Syntax:

```
! comment
```

Examples:

```
! Turn echo off
!
```

Labels and Sections

@ANSWER

The @ANSWER section label marks the script entry point when the script is executed in answer mode.

Syntax:

```
@ANSWER
```

@CCLSCRIPT

Available *only* in Mac OS X.

The `@CCLSCRIPT` section label marks the start of a CCL script. The label is optional and has no functional purpose..

Syntax:

```
@CCLSCRIPT
```

@HANGUP

The `@HANGUP` section label marks the script entry point when the script is executed in hangup mode.

Syntax:

```
@HANGUP
```

@LABEL

The `@LABEL` command sets a numeric label in the script that can then be referenced from other script commands, such as `JUMP`, `JSR`, and `IFTRIES`. A script may include up to 128 labels, numbered 1 through 128. To make debugging easier, assign the labels in ascending sequence. They don't need to be consecutive.

Syntax:

```
@LABEL labelnum
```

Parameter:

```
labelnum
```

A value from 1–128 that specifies the label number.

Example:

```
@LABEL 30
```

@ORIGINATE

The `@ORIGINATE` section label marks the script entry point when the script plays in originate mode (that is, when initiating a call).

Syntax

```
@ORIGINATE
```

CCL Commands

ASK

The `ASK` command causes a dialog box to be displayed to obtain information from the user. The dialog box contains a prompt message, an optional data entry field, a Cancel button, and an OK button. You may need the `ASK` command if your telephone system uses special telecommunications equipment. This command is typically used in originate mode only.

"String Formats" in Chapter 1 shows how to use the `ASK` string as part of another string. The `ASK` string is set if the user presses either the OK button or the Cancel button.

Syntax

```
ASK maskflag "message" [jumpLabel]
```

Parameters

maskflag

0

Echo the user's input/

1

Mask the user's input with bullets (••••).

2

Do not allow user input.

message

The string to display in the dialog box as a prompt for the user.

jumpLabel

If supplied, the label to jump to, where execution continues when the Cancel button is pressed; if not supplied, or if the OK button is pressed, then execution continues at the next CCL line.

Example

```
ASK 1 "Enter your password to access the network."
```

```
ASK 2 "When the remote modem answers, click OK, otherwise click Cancel to stop  
Manual Dialing."
```

CHRDELAY

The `CHRDELAY` command allows you to specify a delay time between characters for all subsequent `WRITE` commands. This is useful for telecommunications equipment that requires data at a speed slower than the interface speed.

Syntax

```
CHRDELAY delay
```

Parameter

delay

The delay time, in tenths of a second.

Example

CHRDELAY 8

COMMUNICATINGAT

For V.32bis devices that support RTS/CTS hardware flow control (including modems with an appropriate cable, as described in “[Cable Specifications](#)” (page 43)), use the `COMMUNICATINGAT` command to indicate the speed of the modem connection if the modem speed is different from the serial port speed. This is necessary because Mac OS X’s internal timers are based on the connection speed.

Syntax`COMMUNICATINGAT baud`*Parameter*

baud

The modem speed, in bits per second.

Example`COMMUNICATINGAT 4800`

DECTRIES

The `DECTRIES` command decreases the variable `tryCounter` by one. The CCL interpreter maintains `tryCounter`, which you may set to a value and increase or decrease by one. See also the commands `IFTRIES`, `INCTRIES`, and `SETTRIES`.

Syntax`DECTRIES`

DTRCLEAR

The `DTRCLEAR` command clears (that is, deasserts) the Data Terminal Ready (DTR) signal on the RS-232 interface.

Syntax`DTRCLEAR`

DTRSET

The `DTRSET` command sets (that is, asserts) the Data Terminal Ready (DTR) signal on the RS-232 interface.

Syntax

DTRSET

EXIT

EXIT terminates execution of the script and returns a result code along with an optional string.

- If the script executes successfully, have it return a result code of 0.
- If the script fails for any reason, it should return the appropriate error result code, as listed in [“Result Codes”](#) (page 41)

To give the user a nonstandard error message, use result code -6002 and use the string parameter to pass the nonstandard error message.

Syntax

EXIT result ["string"]

Parameters

result

One of the CCL result codes listed in Appendix A, "Result Codes".

string

The message displayed to the user when a connection attempt fails; if you include a string for one of the standard result codes, it overrides the message that Mac OS X would normally display.

Examples

EXIT -6022

EXIT -6002 "unable to communicate with PBX"

FLUSH

FLUSH empties all characters from the serial driver input buffer.

Syntax

FLUSH

HSRESET

The HSRESET command sets the serial port's flow control options. If you are using a standard modem cable, you will turn off flow control and leave it off. If you are using a device that supports RTS/CTS handshaking, you need only the outputCTS parameter. Turn all options off at hangup.

Syntax

```
HSRESET outputXON/XOFF outputCTS XON XOFF
inputXON/XOFF inputDTR
```

Parameters

outputXON/XOFF

XON/XOFF handshaking for output. For Mac OS X, it must be off.

outputCTS

CTS hardware handshaking for output. For a modem, if you are using a cable that supports RTS/CTS handshaking, it should be on for originate and answer modes and off for hangup mode.

XON

Specifies the XON character. (DO NOT USE with Mac OS X.)

XOFF

Specifies the XOFF character. (DO NOT USE for Apple Remote Access.)

inputXON/XOFF

XON/XOFF handshaking for input. For Apple Remote Access, it must be off.

inputDTR

DTR hardware handshaking for input. For Apple Remote Access, it should be off. For more information, see *Inside Macintosh*, volume 4 (no longer in print) or *Inside Macintosh: Devices*, available through the Apple Developer Catalog.

Example

```
HSRESET 0 1 0 0 0 0
```

IFANSWER

If the script is executing in answer mode, the IFANSWER command causes execution to continue at the specified label.

Syntax

```
IFANSWER jumpLabel
```

Parameter

jumpLabel

The label to which execution should conditionally jump.

Example

```
IFANSWER 30
```

IFORIGINATE

If the script is executing in originate mode, the IFORIGINATE command causes execution to continue at the specified label.

Syntax

```
IFORIGINATE jumpLabel
```

Parameter

jumpLabel

The label to which execution should conditionally jump.

Example

```
IFORIGINATE 7
```

IFSTR

The IFSTR command compares two strings: one of the variable strings (described in [“Variable Strings \(varStrings\)”](#) (page 17)) and a literal string that you specify in the script. If the strings are equal, the script continues execution at the specified label.

Syntax

```
IFSTR varStringIndex jumpLabel
      "compareString"
```

Parameters

varStringIndex

The number of the variable string to compare.

jumpLabel

The label to which execution should conditionally jump.

compareString

The string to which the variable string is compared.

In the following example, if the modem speaker flag (`varString2`) is on (1), execution jumps to label 55. Otherwise, the next command is executed.

Example

```
IFSTR 2 55 "1"
```

IFTRIES

The IFTRIES command compares a parameter with the variable `tryCounter`. If the value of `tryCounter` is greater than or equal to the parameter, the script continues execution at the specified label. See also the commands `DETRIES`, `INTRIES`, and `SETTRIES`.

Syntax

```
IFTRIES numTries jumpLabel
```

Parameters

numTries

The parameter to compare with the variable `tryCounter`.

jumpLabel

The label to which execution should conditionally jump.

The following example checks to see if the value of `tryCounter` is greater than or equal to 3. If so, execution jumps to label 62 and continues. If not, the next instruction is executed.

Example

```
IFTRIES 3 62
```

INCTRIES

The **INCTRIES** command increases the variable `tryCounter` by one. See also the commands **DECTRIES**, **IFTRIES**, and **SETTRIES**.

Syntax

```
INCTRIES
```

JSR

The **JSR** command causes script execution to jump to the subroutine located at the specific label, saving the address of the line following the **JSR** command. When a **RETURN** command is encountered, execution resumes at the line following the **JSR** command. **JSR** commands can be nested up to 16 deep.

Syntax

```
JSR jumpLabel
```

Parameter

`jumpLabel`

The label where execution should continue after the jump.

Example

```
JSR 50
```

JUMP

The **JUMP** command causes script execution to continue at the specified label.

Syntax

```
JUMP jumpLabel
```

Parameter

`jumpLabel`

The label where execution should continue after the jump.

Example

```
JUMP 59
```

LBREAK

The `LBREAK` command generates a long break (3.5 seconds) on the transmission line.

Syntax

```
LBREAK
```

MATCHCLR

The CCL interpreter has a buffer that holds up to 32 strings loaded by the `MATCHSTR` command. The `MATCHCLR` command erases all strings in the buffer. Use the `MATCHCLR` command before loading each new group of strings. See also the `MATCHREAD` and `MATCHSTR` commands.

Syntax

```
MATCHCLR
```

MATCHREAD

The CCL interpreter has a buffer that holds up to 32 strings loaded by the `MATCHSTR` command. The `MATCHREAD` command reads input from the serial driver and compares the input to the strings currently in the buffer. If a match is found within the specified `MATCHREAD` time, execution continues at the label associated with that match string (as defined by the `MATCHSTR` command that loaded the string). See also the `MATCHCLR` and `MATCHSTR` commands.

Syntax

```
MATCHREAD time
```

Parameter

```
time
```

The time allowed for a match, in tenths of a second.

The following example searches for a match within 3 seconds.

Example

```
MATCHREAD 30
```

MATCHSTR

The CCL interpreter has a buffer that holds up to 32 strings. The `MATCHSTR` command loads a string to the buffer, so that incoming strings can be matched against it by a `MATCHREAD` command. If an incoming string matches the stored string, script execution continues at the label specified in the `MATCHSTR` command. See also the commands `MATCHCLR` and `MATCHREAD`.

Syntax

```
MATCHSTR matchNum matchLabel "matchStr"
```

Parameters`matchNum`

A value from 1–32 specifying which string in the buffer to replace.

`matchLabel`The label where execution should continue when a `MATCHREAD` command detects a matching string.`matchStr`

A string (1–255 characters in length) to compare against.

The following example loads the string "OK\13\10" into the buffer as string 1. If a subsequent `MATCHREAD` reads a string that matches this one, then execution jumps to label 8.

Example

```
MATCHSTR 1 8 "OK\13\10"
```

MONITORLINE

Available *only* in Mac OS X.

Enables or disables Data Carrier Detect (DCD).

Syntax

```
MONITORLINE monitor
```

Parameters`monitor`

0

Disable DCD (soft carrier mode).

1

Enable DCD (hard carrier mode).

The following example enables data carrier detect.

Example

```
MONITORLINE 1
```

NOTE

The `NOTE` command displays status and log information, passing the message string as a parameter. Optionally, you can set the message level to specify where the message should appear.

Note: While in answer mode, NOTE does not write to the activity log or to the status window.

Syntax

```
NOTE msgStr [msgLevel]
```

Parameters

msgStr

The message to display.

msgLevel

The message level (the default level is 3).

1

Send the message to the activity log only.

2

Send the message to the Internet Connect status window only.

3

Send the message to both the activity log and the Internet Connect status window.

The following examples show important places in which you should use the NOTE command. In the first example, the script logs outgoing calls by displaying the dialed phone number in the Internet Connect status window and the activity log. In the second example, the script displays the speed of the connection in the Internet Connect status window.

Examples

```
NOTE "DIALING ^1" 3  
NOTE "Communicating at 9600 bps." 2
```

PAUSE

PAUSE causes script execution to halt for a specified period of time.

Syntax

```
PAUSE time
```

Parameter

time

The time to pause, in tenths of a second.

The following example causes script execution to pause for 2 seconds.

Example

```
PAUSE 20
```

RETURN

The `RETURN` command terminates a subroutine. Script execution continues with the line following the `JSR` command.

Syntax

```
RETURN
```

SBREAK

The `SBREAK` command generates a short break (.5 seconds) on the transmission line.

Syntax

```
SBREAK
```

SERRESET

The `SERRESET` command configures the serial port by passing values for baud rate, parity, data bits, and stop bits to the serial driver. Specifying a value other than the values listed below causes the default value to be used. The defaults for each parameter are listed below.

Syntax

```
SERRESET baudRate, parity, dataBits,  
          stopBits
```

Parameters

baudRate

300, 1200, 2400 (the default), 4800, 9600, 14400, 19200, 28800, 38400, 57600, and so on.

parity

1 for odd parity 2 for even parity 0 or 3 for no parity (the default)

dataBits

5, 6, 7, or 8 (the default)

stopBits

1 for 1 stop bit (the default) 2 for 2 stop bits 3 for 1.5 stop bits

Example

```
SERRESET 9600, 0, 8, 1
```

Note: Apple Remote Access requires no parity, 8 data bits, and 1 stop bit.

SETSPEED

The `SETSPEED` command sets the asynchronous serial interface speed to the specified speed. Use `SETSPEED` to set speeds other than those allowed in `SERRESET`.

Syntax

```
SETSPEED interfacespeed
```

Parameter

```
interfacespeed
```

The serial interface speed.

Example

```
SETSPEED 24000
```

SETTRIES

SETTRIES initializes the tryCounter variable to the specified value. See also the commands DECTRIES, IFTRIES, and INCTRIES.

Syntax

```
SETTRIES tries
```

Parameter

```
tries
```

The value to assign to the tryCounter variable.

Example

```
SETTRIES 0
```

USERHOOK

USERHOOK conveys information about the state of the modem to Mac OS X.

Syntax

```
USERHOOK opcode
```

Parameter

opcode

The user hook to execute.

1

Indicates that the script is answering a call and that a ring is indicated by the modem. This prevents other applications from using the serial port until after the call has terminated.

2

Reports that the modem is doing error correction (other than MNP10, which is indicated by opcode 4).

3

Requests that Mac OS X turn off its built-in data compression.

4

Reports that the modem is doing MNP10 error correction.

Example

USERHOOK 1

WRITE

WRITE writes the specified string to the serial driver.

Syntax

WRITE message

Parameter

message

The message written to the device.

The following example sends to the serial driver the modem command ATDT followed by variable string #1 and a carriage return. (For more information, see [“Variable Strings \(varStrings\)”](#) (page 17).)

Example

WRITE "ATDT^1\13"

Result Codes

This appendix lists the result codes returned by the Communication Command Language (CCL). Each result code is shown with a description of the error and the message, if any, that is displayed to the user.

If the script executes successfully, have it exit with result code 0. If the script is unsuccessful for any reason, have it exit with one of the error result codes listed in this appendix. Note that result code -6002 allows you to pass a custom message to the user.

Result code	Description	Message displayed
-6002	Generic CCL error.	(Supplied by the string parameter in the EXIT command.)
-6003	Subroutine overflow.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6004	The target label is undefined.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6005	Bad parameter error.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6006	Duplicate label error.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6007	Close error.	(No message is displayed.)
-6008	The script was canceled.	(No message is displayed.)
-6009	The script contains too many lines.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6010	The script contains too many characters.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6011	The CCL has not been initialized.	(No message is displayed.)
-6012	Cancel in progress.	(No message is displayed.)
-6013	Another script is in progress.	(No message is displayed.)
-6014	Exit with no error.	(No message is displayed.)

Result code	Description	Message displayed
-6015	A label is out of range.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6016	Bad command.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6017	End of script reached; expected Exit.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6018	The match string index is out of bounds.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6019	Modem error; the modem is not responding.	Modem not responding. Reset modem, check connections, or check to see that the proper port and modem type were specified in the Remote Access Setup control panel.
-6020	No dial tone.	The modem cannot acquire a dial tone.
-6021	No carrier.	The modems could not connect. Try again.
-6022	The line is busy.	The phone number you are calling is busy.
-6023	No answer.	The phone number you are calling does not answer.
-6024	No @ORIGINATE command in the modem script.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6025	No @ANSWER command in the modem script.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6026	No @HANGUP command in the modem script.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.

Cable Specifications

This appendix describes the CTS/RTS handshaking cable that is recommended when using Mac OS X with a V.32bis or faster modem and discusses implications of this wiring scheme for other communications applications.

To make the most efficient use of Mac OS X with a V.32bis or faster modem, use a cable with the specifications shown in Table C-1.

Table C-1 Computer to Modem Cable Specifications

Computer DIN-8	Modem DB-25	Comments
1 (DTR)	4,20 (RTS, DTR)	
2 (CTS)	5 (CTS)	Normally pin 2 (CTS) is connected to pin 6 (DSR) on other cables.
3 (TxD-)	2 (TD)	
4 (SG)	7 (SG)	
5 (RxD-)	3 (RD)	
6 (TxD+)	Not connected	
7 (GPi)	8 (DCD)	
8 (RxD+)	7 (SG)	

Some manufacturers ship their V.32bis and faster modems with a cable that meets these specifications.

Modem Control Issues

A cable constructed as specified in the previous section provides the hardware handshaking that high-speed modems require. If your cable does not meet these specifications, the modem may not operate or may not be able to sustain a connection. The cable supports the following handshaking features:

- CTS handshaking allows the modem to signal the computer to stop sending data to the modem.
- RTS handshaking allows the computer to signal the modem to stop sending data to the computer.
- DTR handshaking allows the computer to signal the modem to reset, hangup the call, or go into command mode.

RTS and DTR cannot be used concurrently. If you want to use RTS, you need to force disconnects by other means than DTR, such as `+++`, `SBREAK`, or `LBREAK`. If you want to use DTR, the computer must be able to accept data at all times. The computer's serial port must be set to a speed equal to or greater than the modem's highest connect speed. The actual connect speed is the modem to modem data rate, rather than the modem's serial port speed. DSR and DCD handshaking are not available with this cable. Therefore other types of communications software, such as terminal emulation software, cannot use DSR and DCD signals to detect modem readiness or carrier presence with this cable.

Recommended Modem Control

The following guidelines provide for optimum performance in most instances:

- Set the computer's serial port speed equal to or greater than the modem's highest connect speed.
- Use CTS handshaking to control data flow to the modem.
- Do not use RTS handshaking.

If possible with your modem type, use DTR control for hanging up and resetting.

Document Revision History

This table describes the changes to *CCL Modem Scripting Guide*.

Date	Notes
2007-06-28	TBD

REVISION HISTORY

Document Revision History