Caché I/O Device Guide

Version 2018.1
2024-04-03

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Caché I/O Device Guide
Caché Version 2018.1 2024-04-03

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Figures

List of Tables

About This Book

This book is a guide to managing I/O and devices with ObjectScript.

Its chapters are:

- Introduction to Caché I/O
- I/O Devices and Commands
- Terminal I/O
- Local Interprocess Communication
- TCP Client/Server Communication
- UDP Client/Server Communication
- Sequential File I/O
- The Spool Device
- Printers
- Magnetic Tape

There is also a detailed Table of Contents.

The following documents provide information about related concepts:

- *The ObjectScript Language Reference*

For general information, see *Using InterSystems Documentation*.

1

About I/O Devices

Caché provides support for many different kinds of devices, both physical devices and logical devices. This book describes how to manage:

- I/O Devices
- Terminal I/O
- Local Interprocess Communication
- TCP Client/Server Communication
- Sequential File I/O
- The Spooler
- Printers
- Magnetic Tape I/O

Because I/O interfaces are often platform-dependent, several chapters in this manual have separate sections for managing an I/O device on different operating system platforms.

This chapter introduces topics related to managing I/O devices in Caché. It includes the following topics:

- Device Management Utilities
- Default Devices
- Identifying Devices
- Defining Devices
- Managing Magnetic Tape Devices
- Accessing Devices
- Defining Default Mnemonic Spaces

Caché supports both physical and logical I/O devices. The supported physical devices include:

- Terminals
- Printers
- Disk drives
- Magnetic tape
- Cartridge tape

The supported logical devices include:

- Principal device
- Spooler
- Sequential files
- Flat files
- Interjob communication (IJC) devices
- Routine interlock devices

1.1 Device Management Utilities

There are several Caché utilities to manage devices. The table below summarizes these utilities and alternative ways of accessing them.

Table 1–1: Caché Device Utilities

Action	Utility	Description
Define devices	Device configuration in the Management Portal	Allows you to define devices users can access with the **%IS** utility. The devices are stored in the *%IS* global. You can edit and delete these device definitions. At the Devices subsection, you define devices, including mnemonics and aliases. Default devices are provided. In the Device Sub-Types subsection, you define device subtypes. Default subtypes are provided. At the Mag Tapes subsection, you associate magnetic tape device definitions (made in the Devices subsection) with the names of magnetic tape drives.
Define default mnemonic spaces	IO configuration options in the Management Portal	You can control devices with the **WRITE** /*mnemonic* command. On this panel, you enter the name of the default mnemonic spaces that Caché uses when executing a **WRITE** /*mnemonic* command that wasn't preceded by an **OPEN** or **USE** command with a mnemonic space specification argument.
Allow users to select a device interactively in character-based applications.	%IS	Programmers call this utility in character-based applications to allow users to select a device by specifying its Caché device mnemonic at a Device: prompt. The Caché device and its mnemonic must be configured using the Devices options from the Management Portal.
Allow users to store print output in a spool file	%SPOOL	For more information, see the chapter "SPOOL Device".

1.2 Default Devices

1.2.1 Devices

When you install Caché, default devices are defined. These are displayed in the **Devices** configuration subsection of the Management Portal. Select **[System] > [Configuration] > [Device Settings] > [Devices]**. View the list of defined devices.

1.2.2 Device Subtypes

Caché ships with many default *device subtypes*. Each device subtype defines device characteristics, such as screen length and form-feed characteristics.

The complete list of subtypes is in the **Device Subtypes** configuration option of the Management Portal. Select **[System] > [Configuration] > [Device Settings] > [SubTypes]**. View the list of defined sub types.

1.2.3 Magnetic Tape Devices

Caché also ships with some magnetic tape devices defined. The complete list of magnetic tape devices is in the **Magnetic Tapes Devices** configuration option of the Management Portal. Select **[System] > [Configuration] > [Device Settings] > [MagTapes]**. View the list of defined magnetic tape devices. Magnetic tape devices can be assigned device numbers 47 through 62; by default, magnetic tape devices are assigned to device numbers 47, 48, 57, and 58. (The magnetic tape devices are also listed in the **Devices** option.)

1.3 Identifying Devices

When you define a device in the **Devices** configuration section of the Management Portal, you provide three *device identifiers* to specify a device:

- *Mnemonic*, which is used at the %IS Device prompt.
- *Device ID*, which is used in an **OPEN** command.
- *Alias*, which can be used in place of a Device ID in an **OPEN** command.

These device identifiers have several advantages over physical device names:

- They uniquely identify logical devices regardless of how many physical devices you have.
- They assign different characteristics to each logical device.
- They allow user applications to reference devices by consistent numbers without having to know the actual physical device names, which may vary on different platforms.

For more information about using device identifiers, see the Accessing Devices section.

1.3.1 Device Mnemonics

You can associate one or more *mnemonics* with a particular Device. You use a mnemonic in response to the "Device:" prompt issued by the %IS character-based utility.

Mnemonics provide these advantages:

- They are flexible, because you can change where the mnemonic points rather than developers having to change their applications.
- They are easy for users and developers to remember. For instance, you can set up a printer device with the mnemonic Printer, or you can set up a Device ID for a file name and give it the mnemonic FILE.

1.3.2 Device IDs

You can identify devices by a number or by their operating system name. You use this identifier in **OPEN** commands.

1.3.3 Device Alias

You can define one or more *alias* values for each Caché device you define. When a user specifies an alias in an **OPEN** command, Caché translates it into the Device ID.

The default Device IDs that Caché provides are appropriate for most users. However, some users may want to override these defaults, particularly for magnetic tape devices. You can do this by providing an alias as part of the device's configuration settings in the Management Portal.

1.3.4 Default Device IDs and Mnemonics

When you install Caché, these are the default device numbers and mnemonics for each type of device.

Table 1–2: Default Device Numbers and Mnemonics

Device	Device ID	Mnemonic	Notes
Principal	0	TERM	You cannot change the Device ID for this device.
Spooler	2	SPOOL	
Magnetic Tape Drive	47, 48	47, 48	
Cartridge Tape Drive	57, 58	57, 58	

Note: Caché also supports device IDs 49-56 and 59-62, but does not initialize these IDs at installation. However, you can add device entries for these IDs, giving them the same value in the mnemonic field, if you have more magnetic tape and/or cartridge tape devices.

However, Caché also recognizes other device numbers that you can use to define devices. The following table lists the recognized default device numbers.

Table 1–3: Caché Default Device Numbers

Device Number	Type	Definition
0	Principal device	For an interactive process, this is the terminal on which the user logs in. For a Caché jobbed process, this is the null device (by default) or the device provided in the argument list for the job command which creates the jobbed process.

Device Number	Type	Definition
1	cconsole.log	Use this device number to send error messages or other special messages to the system console log. For example, issuing the following from a Terminal writes the specified string to the Console Log: `OPEN 1 USE 1 WRITE "This is a test" CLOSE 1`. See also the **WriteToConsoleLog()** method.
2	Caché system spooler	This is a global that stores output so you can direct it to a physical I/O device at another time.
47-62*	Magnetic tapes	Up to 16 tape drives are supported. Only 4 magnetic tap device numbers, 47-48 and 57-58, are defined at installation. You can define the others if you have more than 4 magnetic tape devices.
63	View buffer	Used with the **VIEW** command and **$VIEW** function to transfer data between memory and disk.
20-46, 200-223	Routine interlock devices	Provided for compatibility with DSM locking applications.
4-19, 64-199, 224-255, 2048-2375	IJC devices	Interjob communication (IJC) logical devices. Used to transfer information between Caché processes. You can control the availability of these devices. See the section "Communication Between Caché Processes" for more information.
None	The Null device	/dev/null: the Null device on NL: the Null device on Used to dispose of output you do not want displayed.
256-2047	Terminals, printers, and flat files.	

Note: * Device 50 has a hardcoded blocksize of 2048.

1.3.5 Device Types

In addition to the mnemonics and device numbers, Caché supports I/O *device types*. Each internal device number belongs to one of these types. The following table shows the device types:

Type	Meaning
TRM	Terminal
SPL	Spooling device
MT	Magnetic tape drive
BT	Cartridge Tape drive
IPC	Interprocess communication device
OTH	Any other device, such as a printer

1.4 Defining Devices

You define, edit, and delete devices in the **Devices** configuration settings of the Management Portal. The information you enter is stored in the *^%IS* global. For more information about this global, see the section Structure of ^%IS Global.

If you make device changes while Caché is running, you are prompted as to whether you want to activate the changes without restarting Caché. If you agree to activate the changes, the new definitions are made available immediately to users.

1.5 Managing Magnetic Tape Devices

Caché supports up to sixteen tape drives on each CPU. Any tape drive that your system supports can be used with Caché.

1.5.1 Defining Magnetic Tape Devices

Before you can use a magnetic tape device with Caché, you must define it in the MagTape configuration settings of the Management Portal. Take the following steps:

1. Select **[System] > [Configuration] > [Device Settings] > [MagTapes]**. View the current list of magnetic tape devices.
2. Click **Create New Mag Tape**.
3. Enter the name of the physical magnetic tape device (for example, `\\.\TAPE1`).
4. Enter a device number for the magnetic tape device (within the range 47 through 62).
5. Click **Save** to save the changes.

1.5.2 Deleting Outmoded Assignments

You should delete outmoded magnetic tape device definitions so that your users do not inadvertently select them.

Many magnetic tape utilities (including **%GIF**, **%GOF**, **%RIMF**, and **%ROMF**) ask you whether to wait 15 seconds if the utility is not able to open the magnetic tape or cartridge tape device. If you answer "Yes" or do not respond within 10 seconds, the utility lets you decide whether to continue waiting. If you choose not to wait, the system stops trying to open the device.

1.6 Accessing Devices

On a Caché for Windows system, you must use device numbers for magnetic tape devices, interjob communication devices, and routine interlock devices. For terminals and printers, you can use device mnemonics or device numbers you assign.

On a Caché for UNIX® system, you can use UNIX® file specifications to refer to files or you can set up device numbers to refer to files.

You can access a device in one of two ways:

- Entering a device mnemonic at the "Device:" prompt in the **%IS** utility.
- Issuing an **OPEN** command and entering a Device ID or Alias.

1.6.1 Allowing Users to Select Devices with the %IS Utility

If you want users of a character-based application to select a device interactively, call the **%IS** utility from the application. You can learn more about the **%IS** utility in the section Allowing Users to Specify a Device.

To select a device using the **%IS** utility:

1. At the Device: prompt, enter a device mnemonic.

Table 1–4: Predefined Mnemonic Spaces

Mnemonic	Corresponding Device
<ENTER>	Terminal screen
SPOOL	Spooler
2	Spooler
|PRN|	Default Windows Printer
File name: MYFILE.TXT DEV$:[TEST]MYFILE.TXT C:\MGR\MYFILE.TXT	File at path specified or, if no path specification, in current directory.
47, 48	Magnetic tape device
57, 58	Cartridge tape device

2. Depending on the type of device, you see another prompt:

Table 1–5: Device Utilities

Device	Prompt	Valid Responses
Terminal	Right Margin	A number representing the number of characters per line.
Printer	Right Margin	A number representing the number of characters per line.
Spooler	Name (of file)	A valid file name for the platform, path optional.
Magnetic Tape	Parameters Rewind?	A valid parameter list for an **OPEN** command for the device type.
File Name	Parameters	A valid parameter list for an **OPEN** command for the device type.

1.6.2 Accessing Devices with the OPEN Command

You can use an **OPEN** command either at the Caché programmer prompt or within an ObjectScript application to open a specific device for reading and writing. When you specify the device, you can use its Device ID or its alias.

1.6.3 Interpretation Levels for Devices

Device identifiers you use with %IS or an **OPEN** command go through up to three levels of interpretation. Thus, if you enter the mnemonic 47 at the %IS global "Device:" prompt, the final device ID that is used may be different. The three levels are described below.

1.6.3.1 Level 1: %IS Utility Level

The first level is used if a device is selected with the %IS utility. Mnemonics in the ^%IS global can be associated with device numbers. The %IS utility then issues an **OPEN** command to that device number.

1.6.3.2 Level 2: OPEN Command Level

In an **OPEN** command, Caché checks to see if this number exists in the Alias column of the Device panel table. If so, it translates it to the actual device number or name for that device.

Note: Be sure not to define an alias that matches a Device ID but is associated with a different device if you want to access that device by its mnemonic from ^%IS.

1.6.3.3 Level 3: Magnetic Tape Translation

If the device number is of class Mag Tape, the system checks the **MagTape** configuration settings of the Management Portal to see which physical tape drive device to access.

1.7 Defining Default Mnemonic Spaces

Programmers can control devices by using **WRITE** /*mnemonic* commands in their applications. For instance, programmers can move the cursor to a specific column in the current line on a terminal device when they use the %X364 mnemonic space with this command:

ObjectScript

```
WRITE /CHA(column)
```

The action caused by any particular value of mnemonic is determined by the mnemonic space the **WRITE** command is using. A mnemonic space is a routine with entry points (mnemonics) that define device actions and attributes.

The **WRITE** command uses the mnemonic space defined in the **OPEN** or **USE** command for the device. If the **OPEN** or **USE** command includes no mnemonic space argument, then Caché uses the default mnemonic space for the device type.

For further details on mnemonic spaces, see the.

1.7.1 Predefined Mnemonic Spaces

Caché ships with two predefined (default) mnemonic spaces, described in the following table:

Table 1–6: Predefined Mnemonic Spaces

Routine Name	Description
^%XMAG	Mnemonic space for magnetic tape. Default at startup for magnetic tape devices.
^%X364	Mnemonic space for X3.64 (ANSI) terminals. Default at startup for terminals, sequential files, and all other devices except magnetic tape.

The predefined mnemonic spaces are the default mnemonic spaces for:

- Sequential files (^%X364)
- Magnetic Tape (^%XMAG)
- Terminals (^%X364)
- Other devices (^%X364)

These defaults are defined in the Management Portal. Select **[System] > [Configuration] > [Device Settings] > [IO Settings].**

If you create your own mnemonic space routine(s), you may want to change the default mnemonic spaces Caché uses for one or more of the four device types.

2

I/O Devices and Commands

This chapter describes how to work with I/O devices within Caché applications and at the Caché prompt. It assumes your devices have been set up properly, as described in the About I/O Devices chapter. For additional information about specific devices, see the other chapters in this guide.

- Overview of I/O Commands
- Specifying I/O Devices
- Allowing Users to Specify a Device
- Specifying Devices in I/O Commands
- Processes and Devices
- Application Development I/O Commands
- Device Special Variables
- Controlling Devices with Mnemonic Spaces

2.1 Overview of I/O Commands

The I/O commands allow you to own, use, read from, write to, and close devices. To direct I/O operations to a device, first issue the following commands:

- Issue an **OPEN** command to establish ownership, unless the device is your principal device.
- Issue a **USE** command to make the device the current device.
- Subsequent **READ** and **WRITE** commands read from and write to that device.
- A **CLOSE** command releases ownership of the device so that other processes can use the device.

The following sections give an overview of the Caché I/O commands.

2.1.1 General I/O Syntax

The following general syntax applies to I/O commands that support I/O command keywords in ObjectScript:

```
OPEN device:paramlist:timeout:"mnespace"
USE device:paramlist:"mnespace"
CLOSE device:paramlist
```

where *paramlist* is either a single parameter, or a list of parameters enclosed in parentheses and separated by colons:

```
 parameter  (parameter:parameter[:...])
```

A *parameter* can either be a positional parameter or a keyword parameter. A keyword parameter has the following syntax:

```
/keyword[=value]
```

The leading slash distinguishes a keyword parameter from a positional parameter value. The meaning of a positional parameter value is derived from its position in the colon-delimited list. The meaning of a keyword parameter value is derived from the specified keyword.

Note that both positional and keyword parameters can be specified in the same *paramlist*. For example, the following example mixes positional and keyword parameters to open a new file named test.dat in write/sequential mode with JIS I/O translation:

ObjectScript

```
OPEN "test.dat":("NWS":/IOTABLE="JIS")
```

2.1.2 OPEN Command

OPEN establishes ownership of, and opens an I/O channel to, the device specified. This ownership persists until you issue a **CLOSE** command, your process terminates, or some physical operation closes the device. For physical I/O devices (such as magnetic tape drives) or for interprocess communications (such as TCP connections), this ownership prevents all other processes from accessing the device. For logical I/O devices (such as sequential files), this ownership may allow other processes some form of shared access to the file. The handling of multiple processes that open the same sequential file is highly platform-dependent. Use of the **LOCK** command to restrict access to sequential files is strongly advised.

2.1.2.1 Syntax

```
OPEN device{:{(parameters)}{:{timeout}{:"mnespace"}}}
```

Argument	Description
device	The desired device name, ID number, or mnemonic. The maximum length of *device* is 256 characters.
parameters	*Optional* — One or more parameters specifying additional information necessary for some devices. This parameter list is enclosed in parentheses, and the parameters in the list are separated by colons. The individual parameters are listed in tables in the Interprocess Communications, Magnetic Tape I/O,Sequential File I/O, and Terminal I/O chapters.
timeout	*Optional* — The number of seconds to wait for the request to succeed. The preceding colon is required.*timeout* must be specified as an integer value or expression. If *timeout* is set to zero (0), **OPEN** will make a single attempt to open the file. If the attempt fails, the **OPEN** immediately fails. If the attempt succeeds it successfully opens the file. If *timeout* is not set, Caché will continue trying to open the device until the **OPEN** is successful or the process is terminated manually.
mnespace	*Optional* — The name of the mnemonic space that contains the control mnemonics to use with this device, specified as a quoted string. You can use these control mnemonics with the **WRITE** /mnemonic command when directing I/O to this device.

For further details, refer to the OPEN command in the *Caché ObjectScript Reference*.

2.1.2.2 Examples

These examples show how to use the **OPEN** command on different platforms. They may be typed at the command line or used in routines. When used in routines, you may want to replace platform-specific items with variables.

Examples of OPEN on Windows systems

This command opens outbound Telnet connections from a Windows system to a terminal server:

ObjectScript

```
OPEN "|TNT|node:port"
```

where *node* is the node name and *port* is the IP port on the server.

This command opens an I/O channel to an existing Windows file:

ObjectScript

```
OPEN "c:\abc\test.out":"WS"
```

Example of OPEN on UNIX® Systems

This command opens an I/O channel to the UNIX® terminal device /dev/tty06:

ObjectScript

```
OPEN "/dev/tty06/"
```

2.1.3 USE Command

This command makes the specified device the current device, and sets the special variable *$IO* to that device. To **USE** a device other than your principal device, you must first issue an **OPEN** command for it; otherwise, you will receive a <NOTOPEN> error. Arguments have the same meaning as in the **OPEN** command.

2.1.3.1 Syntax

```
USE device:(args):"mnespace"
```

Argument	Description
device	The desired device name, ID number, or alias. The maximum length of *device* is 256 characters.
args	*Optional* — Additional information necessary for some devices. These are listed in the command keyword tables in the Interprocess Communications, Sequential File I/O, and Terminal I/O chapters.
mnespace	*Optional* — Name of the Caché routine containing the definition of the control mnemonics you can use with the **WRITE** /mnemonic command when directing I/O to this device.

For further details, refer to the USE command in the *Caché ObjectScript Reference*.

2.1.3.2 Examples

These examples show how to use the **USE** command on different platforms. They may be typed at the command line or used in routines. When used in routines, you may want to replace platform specific items with variables.

Examples of USE on Windows systems

This Windows example shows the commands you would use to connect via TCP to a time-of-day server on remote host "larry". It uses the service name `daytime`, which the local system resolves to a port number. The **USE** command replaces the **OPEN** C mode with PSTE mode and turns off any user terminators:

ObjectScript

```
OPEN "|TCP|4":("larry":"daytime":"C")
USE "|TCP|4":(::"PSTE")
```

Examples of USE on UNIX® systems

This UNIX® example shows the commands you would use to open an I/O channel to device "/dev/tty06" and establish it as your current device with the option of using **WRITE** /mnemonic with the X364 terminal mnemonics.

ObjectScript

```
OPEN "/dev/tty06"
USE "/dev/tty06"::"^%x364"
```

2.1.4 READ Command

This command reads data from the current device. For some devices, arguments that begin with asterisks return ASCII numeric information; for others, they indicate control functions.

2.1.4.1 Syntax

```
READ variable:timeout
```

For further details, refer to the READ command in the *Caché ObjectScript Reference*.

2.1.5 WRITE Command

This command writes data to the current device. For some devices, arguments that begin with asterisks let you write ASCII characters using their ASCII numeric values; for others, they indicate control functions. For some devices, arguments that begin with the # character indicate the number of times to write that character.

WRITE /mnemonic syntax allows you to control a device with mnemonics which are defined in Caché code in a mnemonic space. The mnemonic space is a Caché routine that must be made active in an **OPEN** or **USE** command, or configured as a default for the device using the Management Portal. To learn how to define and activate mnemonic spaces, see the section Defining Default Mnemonic Spaces.

2.1.5.1 Syntax

```
WRITE variable
```

For further details, refer to the WRITE command in the *Caché ObjectScript Reference*.

2.1.5.2 Example

To move the cursor to column 1, line 2 on a terminal screen using the predefined ^%X364 mnemonic space, issue the command:

ObjectScript

```
WRITE /CUP(1,2)
```

2.1.6 CLOSE Command

The **CLOSE** command releases ownership of the specified device. **CLOSE** reverses the effect of the **OPEN** command.

2.1.6.1 Syntax

```
CLOSE device[:params]
```

Argument	Description
device	The desired device name, ID number, or mnemonic.
params	The parameter “K” closes the device at the Caché level without closing it at the operating system level. The K parameter has no effect on Windows systems. The file is closed at the operating system level.

If you issue a **CLOSE** command for your principal device, the principal device remains assigned to your process until you log off.

Several other conditions can affect the behavior of **CLOSE**:

- If output to a device is stopped for some reason, Caché may be unable to finish output to that device, in which case you cannot close it, and may not be able to halt. For example, if a terminal sends a **Ctrl-S** to the operating system to tell it to stop output to the terminal, you must resume output to the terminal by pressing **Ctrl-Q**.
- If you close the current device, **CLOSE** changes the value of the system variable $IO to that of the principal device. The **CLOSE** command releases ownership of the current device only after all output to that device is complete.
- When a process halts, the system automatically closes all devices the process opened while in Caché.

  If output to the device is stopped for some reason, Caché may be unable to finish output to that device, in which case you may not be able to close it or be able to halt.

For further details, refer to the CLOSE command in the *Caché ObjectScript Reference*.

2.2 Specifying I/O Devices

When you develop Caché applications or work with I/O devices at the Caché programmer's prompt, there are two ways to specify I/O devices:

- Call the **%IS** utility, which allows you to specify the device by using a mnemonic defined in the *%IS* global.
- Issue the I/O commands **OPEN**, **USE**, and **CLOSE**, using Caché device numbers or operating system file specifications for the devices.

2.3 Allowing Users to Specify a Device

%IS is a general device selection utility for character-based applications. You can use the built-in **%IS** utility to allow users to select a device to which to direct I/O operations. Whenever a device is to be selected, the application program should call the **%IS** utility. This utility allows the user to specify the device to be used and the appropriate **OPEN** command parameters, opens the selected device, then returns device-specific information to the calling program. Users enter a mnemonic that has been defined in the *^%IS* global. **%IS** relies upon IO configuration defaults established in the Management Portal.

This section addresses the following topics:

- How %IS Works
- %IS Mnemonics
- Structure of ^%IS Global

2.3.1 How %IS Works

2.3.1.1 Device Prompt

When you call the **%IS** utility, Caché prompts for a device name. You respond in one of the following ways:

- Enter the desired device name or ID number.
- Enter a mnemonic for the device.
- Press **Return** to select the current device.

%IS responds as follows:

- If you enter a device mnemonic, **%IS** finds the corresponding device in the *^%IS* global and opens it.
- If you enter a device name, **%IS** issues an **OPEN** command to that device.
- If the device is a Caché device ID, **%IS** checks the device table to see if that number is remapped to another actual device number. **%IS** then issues an **OPEN** for the device.

See the discussion "Alternate Devices" that is part of %IS Mnemonics section below for information about using alternate devices.

2.3.1.2 Additional Questions

If the device you specify is a terminal, the utility prompts you with a default right margin. Press **Return** to select that margin or type a different value. If a program later attempts to write past the specified right margin, the operating system inserts a "CR LF" (carriage return and line feed) when the margin is reached. If you select a device other than a terminal, the utility asks other types of secondary questions.

2.3.1.3 Examples

In the following example, the user presses **Return** to specify the terminal. The utility prompts for a right margin, suggesting a default value of 80. At the => prompt the user enters 132 as the new margin setting.

```
%SYS>DO ^%IS
Device: <RETURN>
Right margin: 80 => 132
%SYS>
```

In the following example, the user specifies magnetic tape unit 57. The utility suggests default parameter values, which the user accepts by pressing **Return**. The utility then suggests the rewind default, and the user presses **Return** to accept the default that the tape is not to be rewound at end-of-tape.

```
%SYS>DO ^%IS
Device: 57
Parameters? ("auv":0:2048) => <RETURN>
Rewind? No => <RETURN>
%SYS>
```

2.3.1.4 %IS Sets the Variable IO and Returns Values of Other Variables

When you select a device, **%IS** sets the variable IO to the device name or number used in the **OPEN** command. **%IS** also returns the values of the variables listed in the following table:

Table 2–1: %IS Device Variable Values

Variable	Example	Description
%ANS	Yes	Generic dialog answer.
IO	64	Device number or device mnemonic of selected device.
IOF	#	Form feed. `WRITE #` issues a form feed and changes **$Y**. `WRITE @IOF` should be used to form feed.
IOBS	*8	Backspace. `WRITE $CHAR(8)` issues a backspace and changes **$X**. `WRITE *8` issues a backspace but does not change **$X**. `WRITE @IOBS` should be used to backspace.
IOM	80	Right margin.
IOSL	66	Screen/page length.

Variable	Example	Description
IOT	TRM	Device type.
IOST	C-VT220	Device subtype (VT220 in this example).
IOPAR	("auv":0:2048)	Any other **OPEN** parameters.
MSYS	M/WNT	Type of system (such as UNIX®, Windows NT).
POP	0	If not zero, specifies that no device was selected. That is, the user entered STOP in response to `Device:` prompt.
RMSDF	RW	Read/Write permissions.

2.3.1.5 OPEN Parameters

By default, the **OPEN** command uses the specifications for the device defined in the *^%IS* global. You can override these settings by specifying other settings when you use **%IS**.

2.3.1.6 Issue a USE Command

After running **%IS**, the application must issue a **USE** command to the device opened by **%IS**. You can use the variable *IO*, as long as you understand that its value changes every time you call **%IS**. Then, subsequent Caché I/O commands, such as **READ** and **WRITE**, refer to that device.

2.3.1.7 Issue a CLOSE Command

The user or application developer must close devices opened with the **%IS** utility.

2.3.2 %IS Mnemonics

%IS has several features to simplify its use. For example, if you want to send I/O to your own terminal, simply press **Return** at the "Device" prompt. You can also use built-in default mnemonics or new mnemonics you define yourself.

2.3.2.1 Device Mnemonics

It is useful to have mnemonics for the various devices and, in some cases, to have more than one mnemonic for a single device. Multiple mnemonics allow you to specify different device characteristics for the device and vary characteristics according to the manner in which the device is used. For example, a terminal that is normally used for data entry, and thus has the characteristics of a terminal, may have an auxiliary printer attached. By assigning a different mnemonic that opens the same device with different characteristics, you can treat the terminal/printer combination as a printer when you want hard copy.

You can configure device mnemonics and characteristics using the Management Portal. To learn how to define and activate mnemonic spaces, see the section Defining Default Mnemonic Spaces.

2.3.2.2 Default Mnemonics

The *^%IS* global is initialized at installation with several default mnemonics. For instance, there are two default mnemonics, SPOOL and 2, for the Caché spooler. Simply type "2" or "SPOOL" to send output to the Caché spooler.

If you are logged in on an RT:, LT:, or VT: type device, and your terminal is the current device, **%IS** will accept 0, "", or the value of *IO* in response to the "Device" prompt. It will use the appropriate template (RT0:, LT0: or VT0:) for your terminal type to generate the information for your terminal.

2.3.2.3 Alternate Devices

If users enter an "A" at the Device prompt, output goes to the alternate device defined for the current device. Usually, users expect the alternate device to be a printer. Instead of defining a separate alternate device for each device in the system, you can create a device, pointing to a printer, with the mnemonic "A". Then, when users enter "A" at the **%IS** "Device" prompt, output goes to that device.

2.3.2.4 CURRENT^%IS Entry Point

CURRENT is an internal entry point within the **%IS** utility that you can use to obtain the device parameters of the current device. This call to **%IS** returns the values of different variables, so you can keep one set of parameters for your principal device and a different set for a device with different characteristics. Ordinarily, you make a call to this internal entry point when you log in, to allow the application access to the device characteristics of the principal device. **CURRENT^%IS** returns the values of the variables listed in the table below:

Table 2–2: CURRENT Return Values

Variable	Example	Description
FF	3	`WRITE @FF` should be used for form feed on this device
BS	*8	`WRITE @BS` should be used to backspace
RM	80	Right margin
SL	24	Screen/page length
SUB	C-VT100	Device subtype
XY	(see Example below)	Set **$X** to DX and **$Y** to DY to perform direct cursor positioning

2.3.2.5 Example

After calling **CURRENT^%IS**, set **$X** and **$Y** to DX and DY to position the cursor.

ObjectScript

```
DO CURRENT^%IS
WRITE *27,*61,*DY+32,*DX+32
SET $X=DX,$Y=DY
```

2.3.2.6 IN^%IS Entry Point

IN is an internal entry point within **%IS** that can be called by routines that only plan to do input from the device. This entry point can be used to ensure that you do not select an output-only device such as a printer.

```
%SYS> Do IN^%IS

Device: 3
Right margin: 132= <RETURN>
[you can't read from this device]
Device: <RETURN>
Right margin: 80= <RETURN>
%SYS>
```

2.3.2.7 OUT^%IS Entry Point

OUT is an internal entry point within **%IS** that can be called by routines that only plan to do output to the device. This entry can be used to check such conditions as whether a magnetic tape is write enabled.

```
%SYS>DO OUT^%IS

Device: 47
Parameters: ("AUS":0:2048)
Rewind? No= Y
[Tape is write locked]
Device:
%SYS>
```

2.3.2.8 Spooling

Caché spooling is independent of the spooling performed by your operating system. Spooling in Caché is a technique that lets you automatically save the output of a program in a global instead of printing it immediately. You can print the output later by sending the contents of the global to the printer.

The mnemonic SPOOL is a default mnemonic. To specify spooling, enter "SPOOL" in response to the Device prompt. The system then asks for a spool file name and description. This is a named used in the ^*SPOOL* global—not a separate file name at the operating system level.

If any existing file names start with or match the name you specify, they are displayed, and you are asked to choose one. If you select none of the existing files, the system allows you to create a new file with the specified name and description as shown in the following example:

```
Device: SPOOL
Name:TEST
1. 1 TEST 02 Nov 1999 10:17 am First test
2. 2 TEST 02 Nov 1999 10:18 am Second Test
Select one: <Return> not found
Create new document 'TEST'? Yes => yes
Description: Third Test
```

If you reselect an existing document because you would like to continue adding to an existing file, the system gives you the following options:

1. Add to the very end of the file;
2. Restart at the top of the last page, in which case the lines that will be deleted are displayed on the screen;
3. Restart at page 1 (the beginning).

You can pass the variables listed in the table below to **%IS** when you call it for spooling.

Table 2–3: Spool Variables You Can Pass to %IS

Variable	Function
IODOC	Document name (when this variable exists and is not a null string all questions are suppressed, and a new document with this name is automatically created).
IODES	Free text description.
IOPGM	Name of a routine that should be called at print time to allow the user to set up printer for the proper forms alignment.

2.3.2.9 Further Features of %IS

%IS can also be used to perform the following tasks:

- Right margin suppressing—It is possible to set up a terminal line so that whenever that device is selected, the Right margin question is suppressed; the default value is automatically assumed.
- Automatic device selection—If the variable *IOP* exists when the **%IS** utility is called, the utility automatically tries to open that device rather than ask for a device. If **%IS** is unsuccessful, it sets the variable *POP* to 1.
- Preconfigured terminals—Using the Management Portal, you can configure a device that does not request any device information from the user.

2.3.3 Structure of ^%IS Global

The *%IS* global is stored in the %SYS namespace. It contains two subscripts. The first subscript is the mnemonic name configured for the device in the Management Portal. Select **[System] > [Configuration] > [Device Settings] > [IO Settings]** to display the default mnemonic for different device types. The second subscript can be 0 or 1.

2.3.3.1 Contents of Node 0

Node 0 contains the Device panel Location value:

```
^%IS(mnemonic,0) = Location
```

In this example, the device with the mnemonic name 47 was given the Device panel Location of "Magnetic tape":

```
^%IS(47,0) = Magnetic tape
```

2.3.3.2 Contents of Node 1

Node 1 contains the other Device panel field values separated by a caret (^):

```
^%IS(mnemonic,1) = Device #^Type^Subtype^Prompt code^not used
^Other Open parameters^Alternate device
```

In this example, the device with the mnemonic name 2 (which is a default name for the Caché spooler) has a device number of 2, device type of SPL (spool), device subtype of PK-DEC. The other values are not defined for a spool type device.

```
^%IS(2,1) = 2^SPL^PK-DEC^^^^^
```

2.4 Specifying Devices in I/O Commands

When you use the I/O commands **OPEN**, **USE** and **CLOSE** to process I/O on any device other than the one on which you are working, you must specify an I/O device. You can specify devices in one of three ways, depending on device type, as shown in the table below.

Table 2–4: Specifying a Device in an I/O Command

Type of Specification	Use for these Devices
Caché Device Name	Terminals and Printers
Caché Device ID or Device Alias	All devices except sequential files
File Name	Sequential Files

Note that Windows and UNIX® handle printer I/O differently. For details, refer to the Printers chapter of this manual.

2.4.1 Specifying Terminals and Printers by Device Name

If your I/O operations are to terminal (or a printer on some platforms), you can use the device name applied by the operating system (UNIX® or Windows) to specify the device. The form is as follows:

```
OPEN "device"  USE "device"  CLOSE "device"
```

Parameter	Description
device	The operating system name of the device, enclosed in quotes. The maximum length of *device* is 256 characters.

2.4.1.1 Specifying a Terminal on Windows Systems

To open an I/O device connected to a serial communications port, specify an **OPEN** command with the following syntax:

```
OPEN "comn:"
```

where *n* represents the number of the port to which the device is attached.

Parameter	Description
n	The number of the port to which the device is attached.

ObjectScript

```
OPEN "com1:"
```

2.4.1.2 Specifying Terminals and Printers on UNIX®

To open an I/O device on a terminal that has the UNIX® device name /dev/tty06, enter the following command

ObjectScript

```
OPEN "/dev/tty06"
```

On UNIX® systems, a printer is identified by the name on the **OPEN** command and is handled as a “character special” file on a tty device. Thus the **OPEN** and **USE** command arguments supported are the same as those for terminal I/O, *not* sequential file I/O. On Windows systems, printer I/O *is* handled like sequential file I/O.

2.4.2 Specifying Devices by Caché ID

For compatibility with other InterSystems products and for convenience, you can refer to devices by device numbers (which are stored in the device table). The system manager can link these numbers to devices using the Management Portal. Select **[System] > [Configuration] > [Device Settings] > [Devices]** to create a new device or edit an existing device.

The system manager can also cause a translation from one number to another. Thus, you can issue an `OPEN 47` and Caché will translate it to `OPEN 49`.

CAUTION: You must use Caché device numbers when referring to magnetic tape devices.

The following table shows the device numbers.

Table 2–5: Caché Device Numbers and Devices

Device Numbers	Devices
0	Principal device (the device on which you logged in).
2	Caché spooler. UNIX®: the mnemonic SPOOL applies to this device.
3	An invalid device number. Attempting to open it returns a <NOTOPEN> error without waiting for *timeout* expiration.
47-60	Magnetic tape devices.
63	View buffer.
20-46, 200-223	Routine interlock devices.
224-255	Interjob communication devices.

2.4.2.1 Examples

To open a magnetic tape device, you issue a command such as:

ObjectScript

```
OPEN 47
```

To open the spooler, you issue the command:

ObjectScript

```
OPEN 2
```

2.4.3 Specifying Files on Disk

You can open a disk file using the operating system file specification enclosed in double quotes.

A Windows file specification has the following format:

```
device:\directory\file.type
```

A UNIX® file specification has the following format:

```
/directory/name
```

For further details, refer to Specifying a File in the "Sequential File I/O" chapter of this manual.

2.4.3.1 UNIX Examples

If your current default directory on a UNIX® or Windows system is /usr/user, you can open a file named pat_rec.dat stored in your current default directory by specifying:

ObjectScript

```
OPEN "pat_rec.dat"
```

The system opens the file automatically. For a new file, include the parameter string "WN" to avoid a hang.

To open a file with the same name, pat_rec.dat, stored in another directory, you must also specify the directory, as follows:

ObjectScript

```
OPEN "/usr/elsewhere/pat_rec.dat"
```

2.5 Processes and Devices

2.5.1 Principal Device and Current Device

2.5.1.1 Each Process has a Principal Device

Each Caché process has one principal input device and one principal output device. By default, these are the same device. When you log in at a terminal and activate Caché, that terminal becomes your principal device. Because Caché implicitly issues **OPEN** and **USE** commands for that terminal, you can issue **READ** and **WRITE** commands to it immediately. The Caché principal device is the one that your operating system has assigned as your principal input device. The **$PRINCIPAL** special variable contains the device ID of the principal device.

2.5.1.2 Caché Directs I/O Commands to the Current Device

Caché directs input and output operations, including **READ**, **WRITE**, **PRINT**, and **ZLOAD** commands, to your current device. Your process' **$IO** special variable contains the device ID of your current device. When you log in to Caché at a terminal, **$IO** initially contains your terminal's device name. In other words, your principal device and your current device are the same immediately after you log in. After you issue a **USE** command, your current device (the one contained in **$IO**) is normally the one named in the last **USE** command you executed.

Although you may issue **OPEN** and **USE** for a device other than your principal device in programmer mode, each time Caché returns to the "**>**" prompt, it implicitly issues **USE** 0. To continue using a device other than 0, you must issue a **USE** command in each line you enter at the "**>**" prompt.

2.5.1.3 When Your Principal Device Becomes Your Current Device

Your principal device automatically becomes your current device when you do any of the following:

- Sign on for the first time.
- Issue a `USE 0` command.
- Issue a call to the **ChangePrincipal()** method of the %Library.Device class.
- Cause an error when an error trap is not set.
- Close the current device.
- Return to programmer mode.
- Exit Caché by issuing a **HALT** command.

2.5.1.4 USE 0 Opens the Principal Device

`USE 0` implies an **OPEN** command to the principal device. If another process owns the device, this process hangs on the implicit **OPEN** as it does when it encounters any **OPEN**.

Issuing a **USE** command for any other device that the process does not own (due to a previous **OPEN** command) produces a <NOTOPEN> error.

An **OPEN** command with no timeout returns control to the process only when the process acquires the device. You can interrupt the open command by a keyboard interrupt command like **Ctrl-C**. An **OPEN** that cannot succeed because of a protection problem or an invalid device name hangs forever. When you specify a timeout in the **OPEN** command, the **OPEN** returns control to your process when the timeout expires.

2.5.2 The Null Device

2.5.2.1 Use the Null Device to Redirect I/O

If your application generates extraneous output which you do not want to appear on your screen, you can direct that output to the null device. You specify the null device by issuing a Caché **OPEN** command with the appropriate argument (see table). Caché treats it as a dummy device.

Table 2–6: Null Device Arguments

Platform	Null Device Argument
UNIX®	/dev/null/
Windows	//./nul

Subsequent **READ** commands immediately return an empty string. Subsequent **WRITE** commands immediately return success. No actual data is read or written. The NULL device bypasses UNIX® open, write, and read system calls entirely.

Note: If you open the NULL device other than from within Caché (for example, by redirecting Caché output to /dev/null from the UNIX® shell), the UNIX® system calls do occur as they would for any other device.

2.5.2.2 Jobbed Processes Use the Null Device

When one process starts another with the **JOB** command, the default principal input and output device of the jobbed process is the null device.

2.5.3 One Process Owns a Device

Only one process can own a device at a time, except sequential files.

In other words, after a process successfully issues an **OPEN** command for a device, no other process can open that device until the first process releases it. A process releases the device in any of the following ways:

- By explicitly issuing a **CLOSE** command.
- By halting.

2.6 Application Development I/O Commands

There are a special set of I/O commands to load, edit, print, and save Caché routines. These commands load routines from and save them to the current device; they are summarized in the table below

Table 2–7: Application Development I/O Commands

Command	Description
ZLOAD [routine]	The **ZLOAD** command, without arguments, loads a Caché routine from the current device. You can use **ZLOAD** with **OPEN** and **USE** to output or input routines from different devices. **ZLOAD** ends when it receives a null line from terminal input or reaches the end of the file.
PRINT [args] *or* ZPRINT [args]	Prints the routine in memory to the current device. It writes an empty line after the last line of the routine. Optional arguments let you control the number of lines you print.
ZSAVE [routine]	**ZSAVE** writes the routine in memory back to disk, giving it the name you supply. If you do not provide a name, it uses the name of the routine you loaded with **ZLOAD**.

2.7 Device Special Variables

Some I/O commands affect the value of certain system variables. This section defines these variables and tells why you might want to use them. These variables are changed only when an I/O command is issued to the current device. These device special variables are summarized in the table below:

Table 2–8: Device Special Variables

Variable	Purpose
$IO	Contains the device ID of the current device, to which all output operations are directed. Caché sets the value of *$IO* to the principal output device at login, and only the **USE** and **CLOSE** commands, a **BREAK** command, or a return to programmer mode can change this value.
$X	Contains a running total of printable characters written since the last carriage return on the current device. This number ranges from 0 to the width of the device.
$Y	Contains a running total of line feeds written since the last form feed on the current device. This number ranges from 0 to the length of the device.
$ZA	Contains **READ** status information after a **READ** command to a terminal device.
$ZB	Contains the character sequence or event ended the last **READ** operation on the current device.
$ZMODE	Contains the parameters you used with the **OPEN** or **USE** command for the current device.

$X and **$Y** are useful in formatting printed output. For more information on them, see the chapter "Terminal I/O." See individual chapters of this document for device-specific information about **$ZA** and **$ZB**.

2.8 Controlling Devices with Mnemonic Spaces

A mnemonic space is a Caché routine that performs device control actions, such as cursor movement and device attributes. Each action is associated with a label. These labels are the mnemonics used in the **WRITE** /mnemonic command. For more information on the **WRITE** /mnemonic syntax, see the **WRITE** command description for each device type in the other chapters of this document.

2.8.1 Predefined Mnemonic Spaces

Caché provides predefined mnemonic spaces described in the table below.

Table 2–9: Predefined Mnemonic Spaces

Routine Name	Device Type Default	Description
^%MAGTAPE	Magnetic tape	Mnemonic space for magnetic tape. For information, see "Magnetic Tape Mnemonic Space for `WRITE /mnemonic`".
^%X364	Terminals, Sequential files, Other devices	Mnemonic space for X3.64 (ANSI) terminals. For information, see "Mnemonic Space for X3.64".
^%XDTM	DTM PC Console	Mnemonic space for DTM PC Console. For information, see "Mnemonic Space for DTM PC Console".

2.8.1.1 Set Up Default Mnemonic Spaces

You can change the mnemonic space that is a default for the following device types in the Management Portal. Select **[System] > [Configuration] > [Device Settings] > [IO Settings]**. This displays the mnemonics for the following:

- Terminals
- Magnetic tape
- Sequential files
- Other

After a default mnemonic space is defined, the control mnemonics in the default mnemonic space for the current device are used if a `WRITE /mnemonic` command is issued, unless the default mnemonic space is overridden by a mnespace argument to the **OPEN** or **USE** command for the current device.

2.8.2 Creating a Mnemonic Space

You can create your own mnemonic space routines. For example, you might want to use a predefined mnemonic space for magnetic tape but create your own for terminal I/O.

1. Create a Caché routine containing the control mnemonics you want. Keep in mind the following points about your routine:
   - The entry points in this routine must be uppercase. These entry points are the mnemonics you reference in **WRITE** /mnemonic commands.

- Some entry points may require arguments. The code in the mnemonic space at an entry point performs an action on the current device.
- Cursor movement routines do not move the cursor past the edge of the screen in any direction, nor do they wrap the cursor.

2. To make this mnemonic space available to all users, give the Caché routine a name that begins with "%" and put it in the system manager's namespace (%SYS).

2.8.3 Select a Mnemonic Space

Before you issue `WRITE /mnemonic` commands to a device, you decide whether you want to use the default mnemonic space for the device type as specified in the Management Portal configuration setting.

- When using the default mnemonic space, do not include a mnespace parameter when you issue **OPEN** or **USE** commands for the device.
- To use another mnemonic space, specify its name in the mnespace parameter of the **OPEN** or **USE** command you issue for the device.

ObjectScript

```
USE "device"::"^%X364"
```

For information on using the *mnespace* parameter, see the OPEN command and the USE command, as well as the chapters on individual device types.

3

Terminal I/O

This chapter discusses terminal I/O in Caché.

3.1 Overview of Terminal I/O Capabilities

ObjectScript provides commands that support serial asynchronous ASCII terminals. You can also use these commands with console I/O.

Using Terminal I/O, your routine can:

- Enable or disable the echo of incoming characters.
- Send and receive ANSI-standard escape sequences.
- Control keyboard interruptions and program special user interactions, including formatted screens, reverse video, and special keys for skipping fields.
- Enable and disable recognition of **Ctrl-C** interrupts.
- Control the flow of incoming and outgoing data by XON (**Ctrl-Q**) and XOFF (**Ctrl-S**).
- Specify COM port state parameters and modem baud rate.
- Conform to foreign protocols when you specify your own set of termination characters.
- Communicate with non-terminal devices, such as automated instruments.

Printers are handled as terminal I/O devices on most platforms. UNIX® systems always handle a printer as a terminal I/O device. On Windows, a printer connected through a serial communications port is handled as a terminal I/O device. Otherwise, Windows systems handle printers as sequential file I/O devices. For further details, refer to the Printers chapter in this manual.

3.1.1 Your Login Terminal or Console is Your Principal Device

The terminal or console on which you log in to Caché is your principal device. You need not open your principal device. If you have not issued an **OPEN** and a **USE**, the first time a process issues a **READ** or **WRITE**, the system opens your principal device automatically, and establishes it as the current device, as if you had issued `OPEN 0 USE 0` explicitly.

Note: Through the rest of this chapter the word *terminal* is used to refer to both terminals and consoles.

3.2 Special Variables Show I/O Conditions

I/O commands can affect the values of special variables. You can test these variables to determine I/O conditions:

- $IO contains the name of the current device.
- $TEST contains a boolean value that shows whether the most recent timed operation was successful.
- $X and $Y show the position of the cursor.
- $ZA, $ZB, and $KEY show information about **READ** operations. **$ZB** and **$KEY** are similar, but not identical.

See the chapter "I/O Devices and Commands" for more information on the device-independent **$IO** special variable. The next sections describe terminal-specific information about the remaining special variables.

3.2.1 $X and $Y and Cursor Position

$X contains the horizontal position and **$Y** the vertical position of the cursor or print head. `$X=0,$Y=0` denotes the upper left corner of the CRT screen or the printed page. Caché calculates both **$X** and **$Y** modulo 256; that is, they range from 0 to 255 and then begin again at 0.

The following table shows the effects of writing or echoing the characters

Table 3–1: Effects of Echoing Characters

Character	ASCII Code	Effect on $X	Effect on $Y
Form Feed	12	$X=0	$Y=0
Return	13	$X=0	$Y=$Y
Line Feed	10	$X=$X	$Y=$Y+1
Backspace	8	$X=$X-1	$Y=$Y
Tab	9	$X=$X+1	$Y=$Y
Any printable ASCII character	32 through 126	$X=$X+1	$Y=$Y

The S protocol of **OPEN** and **USE** turns off the echo. This protocol also disables the changing of **$X** and **$Y** during input, so that they truly indicate the cursor's position.

3.2.1.1 WRITE * and $X and $Y

WRITE * does not change **$X** and **$Y**. Thus, you can send a control sequence to your terminal and **$X** and **$Y** will still reflect the true cursor position. Some control sequences do move the cursor, so you can set **$X** or **$Y** directly when you need to.

3.2.1.2 $X and $Y Example

In the following example, a control sequence moves the cursor in a VT100 terminal to line 10, column 20, and sets **$X** and **$Y** accordingly.

ObjectScript

```
 ; set DY and DX to desired
 ; values for $Y and $X
 SET DY=10
 SET DX=20
 ; ...
 ; escape sequence moves
 ; cursor to desired position
 WRITE *27, *91, DY+1, *59, DX+1, *72
 ; ...
 ; updates $X and $Y
 SET $Y=DY
 SET $X=DX
```

3.2.1.3 Effect of Escape Sequences on $X and $Y Varies

Escape sequences can alter the effect of echoing on the values of **$X** and **$Y**. Three factors control this effect:

- Your operating system, which sets the default behavior.
- Whether /NOXY (which disables **$X** and **$Y** processing) was specified in the **OPEN** or **USE** command.
- You can set how **$X** handles escape sequences for the current process using the **DX()** method of the %SYSTEM.Process class. The system-wide default behavior can be established by setting the *DX* property of the Config.Miscellaneous class.

Escape Sequences Affect $X and $Y on Windows and UNIX® Systems

By default on UNIX® and Windows, when writing or echoing any string that includes the ASCII Escape character (decimal value 27), Caché updates **$X** and **$Y** just as it does for any other character sequence. Thus, ANSI standard control sequences, which the terminal acts on, but does not display, can upset the relationship of **$X** and **$Y** to the cursor's position.

The easiest way to avoid this problem is to use the **DX()** method to alter the behavior (see the next section). Alternatively, you can use the ASCII value of each character in the string in a `WRITE *` statement.

Control Sequence Example

Instead of using the code:

```
%SYS>WRITE $CHAR(27)_"[lm"
```

you can use the following equivalent statement that does not update **$X** and **$Y**:

```
%SYS>WRITE *27,*91,*49,*109
```

Switches Control Updates of $X for Escape Sequences

To select non-default behavior for updating **$X** for your process, issue the **DX(*n*)** method of the %SYSTEM.Process class.

The system manager can alter the system-wide default behavior by setting the *DX* property of the Config.Miscellaneous class.

In both cases, *n* has a value from 0 through 4, as follows:

Value	Default Behavior for Updating $X
0	Caché for UNIX® and Windows behavior (default on those systems)
1	DSM behavior
2	DTM/MSM behavior

For more information, see $X in the *ObjectScript Language Reference*.

3.2.2 $TEST Shows Timed Operation Results

The **$TEST** special variable is set by commands that take a timeout value. These commands include **OPEN** and **READ**. The value of **$TEST** can be set to 1 or 0:

- **$TEST** is set to 1 if the timed command succeeded before the timeout expired.
- **$TEST** is set to 0 if the timeout expires on a timed command.

Note: **OPEN** and **READ** commands without a timeout have no effect on **$TEST**.

For more information, see $TEST in the *ObjectScript Language Reference*.

3.2.3 $ZA Shows READ Status

The **$ZA** special variable contains a number of bit flags that show the status of the last **READ** on the current device. You cannot set **$ZA**; Caché controls its value. **$ZA** remains valid until the next **READ**. **$ZA** contains the sum of the values listed in the table, which shows how your program can test this variable. (**$ZA** also contains bit flags for modem connection status, which are not listed here. For a complete list of $ZA bit flag values, see $ZA in *ObjectScript Language Reference*.)

Table 3–2: $ZA Read Status Values

Value	Test	Meaning
1	$ZA#2	A **Ctrl-C** arrived, whether or not breaks were enabled.
2	$ZA\2#2	The **READ** timed out.
256	$ZA\256#2	Caché detected an invalid escape sequence.
512	$ZA\512#2	The hardware detected a parity or framing error.

While many of the conditions that **$ZA** shows are errors, they do not interrupt the program's flow by trapping to the **$ZTRAP** special variable. Programs that are concerned with these errors must examine **$ZA** after every **READ**. Of course, a **Ctrl-C** with breaks enabled will trap to **$ZTRAP**. For more on error trapping and **$ZTRAP**, see the Error Processing chapter of *Using Caché ObjectScript* and $ZTRAP in the *ObjectScript Language Reference*.

3.2.4 $ZB Shows What Ended a READ

$ZB shows what character sequence or event ended the last **READ** operation on the current device. You cannot set **$ZB**; Caché sets the value of **$ZB** each time you perform a **READ**. You can use this value to act on non-printable characters such as the up arrow key or function keys.

$ZB can contain any of the following:

- A termination character, such as a carriage return

- An escape sequence
- Character number *y* of a fixed-length `READ x#y`
- The single character of `READ *x`
- An empty string after a timed **READ** expires

$ZB never contains more than 64 characters. A longer escape sequence is invalid.

3.2.4.1 $ZB Example

The following example assigns the user-specified input characters to the **READ** command variable *x*, and assigns the input terminator (usually the Return character) to the **$ZB** special variable. When issuing this command from the terminal prompt, you need to set a variable to trap the value of **$ZB** on the same command line as the **READ** command. This is because the line return used to issue a command line is written to **$ZB** as a terminator character. This example uses **ZZDUMP** to display the value of the characters trapped by **$ZB**.

```
USER>READ x SET y=$ZB
USER>ZZDUMP y

0000: 0D
USER>
```

3.3 OPEN and USE Commands

3.3.1 OPEN Command

Establishes ownership of the terminal. An optional parameter list can set the right margin, specify device protocols, and specify one or more termination characters. Following the parameter list, you can optionally specify a *timeout* argument, and/or a *mnespace* argument. The *mnespace* argument specifies the Caché routine where control mnemonics for use with `WRITE /mnemonic` are defined.

OPEN pauses the process until the system finishes opening the device. If you press **Ctrl-C** to interrupt the **OPEN** command, a <NOTOPEN> error results.

OPEN retains control until the opening of the device is complete, unless you specify a timeout. With a timeout, if Caché cannot open the device in the number of seconds you specify, it sets **$TEST** to 0 and returns control to the process. Even if a device is unavailable at the operating-system level, **OPEN** keeps trying to obtain the device until it succeeds or the timeout expires.

3.3.1.1 OPEN Syntax

The **OPEN** command takes the following arguments:

```
OPEN terminal:(margin:protocols:terminator:portstate:baud):timeout:"mnespace"
```

Only the *terminal* argument is required. The *terminal* argument can be an expression whose value is the name of a terminal device. Zero (0) is the process's principal device. **$IO** is the current device. The maximum length of *terminal* is 256 characters.

Arguments are separated by colons (:). If you omit an argument within the list, you must specify the colon as placeholder. However, trailing colons are not permitted; you must not end either the command or its parameter list with a colon.

The optional parameter list is enclosed in parentheses and can contain the following optional parameters:

- *margin* is an integer that specifies the number of characters per line by specifying the right margin.
- *protocols* is one or more letter codes that specify terminal options.
- *terminator* is a string of one or more characters that terminate a **READ** operation. These characters supplement the termination characters that are defined for a specific *protocols*.
- *portstate* is a string that specifies the COM port state.
- *baud* is an integer that specifies the baud rate for a COM port.

You can specify these optional parameters as either positional parameters (in the order shown), or as keyword parameters with the syntax /KEYWORD=value. Keyword parameters may be specified in any order; because Caché executes parameters in left-to-right order, interactions between parameters may dictate a preferred order in some cases. You can mix positional parameters and keyword parameters in the same parameter list. The enclosing parentheses are required if you specify more than one parameter.

The following parameter lists are equivalent:

ObjectScript

```
OPEN $IO:(80:"BFU":$CHAR(13))
 ; all positional
OPEN $IO:(80::$CHAR(13):/PARAMS="BFU")
 ; mixed positional and keyword, using the /PARAMS keyword
 ; to specify a protocol letter code string.
OPEN $IO:(/MARGIN=80:/TERMINATOR=$CHAR(13):/BREAK:/FLUSH:/UPCASE)
 ; all keyword, using separate keywords
 ; for each protocol letter code.
```

Following the parameter list (or a placeholder colon, if no parameter list is specified), you can specify an optional *timeout* in seconds, and a *mnespace* argument to specify the routine that contains the control mnemonics for this device.

For more information, see OPEN in the *ObjectScript Language Reference*.

3.3.2 USE Command

Makes the specified terminal the current device. In programmer mode, all subsequent I/O commands on the same line of code refer to that device. In application mode, the device you name in a **USE** command remains the current device until the next **USE** command.

3.3.2.1 USE Syntax

The **USE** command takes the following arguments:

```
USE terminal:(margin:protocols:terminator):"mnespace"
```

The *terminal* argument can be an expression whose value is the name of a terminal device. Zero (0) is the process's principal device. **$IO** is the current device. The maximum length of *terminal* is 256 characters.

Arguments are separated by colons (:). If you omit an argument, you must specify the colon. You must not end either the command or its parameter list with a colon.

The optional parameter list is enclosed in parentheses and can contain the *margin*, *protocols*, and *terminator* parameters. You can specify the optional *margin*, *protocols*, and *terminator* parameters as either positional parameters (in the order shown), or as keyword parameters with the syntax /KEYWORD=value. Keyword parameters may be specified in any order; because Caché executes parameters in left-to-right order, interactions between parameters may dictate a preferred order in some cases. You can mix positional parameters and keyword parameters in the same parameter list. The enclosing parentheses are required if you specify more than one parameter.

To specify COM port state and baud rate with the **USE** command, use the appropriate keyword parameters.

Following the parameter list (or a placeholder colon, if no parameter list is specified), you can specify an optional *mnespace* argument, which identifies an ObjectScript routine where control mnemonics for use with `WRITE /mnemonic` are defined.

For more information, see USE in the *ObjectScript Language Reference*.

3.3.3 Positional Parameters for OPEN and USE Commands

The following positional parameters are available for the **OPEN** and **USE** commands. You can set these parameters for a device in either the **OPEN** or **USE** command, or take the defaults configured in the Management Portal. These parameters are positional; if you omit a parameter, you must include its preceding colon as a placeholder.

3.3.3.1 margin

The 1st positional parameter: An integer value specifying the right margin (and thus the number of characters per line). Values from 1 to 255 set the right margin for output; any other value disables the right margin. An empty string leaves the margin setting unchanged. On Windows platforms, you cannot use “:n” to control the print margin used. Such notation is ignored by Caché. Code such as “|PRN| :121” is ignored. To control the printer width, send the appropriate control characters for that printer. The notation does work on other platforms.

The default margins for various terminal types are defined in the Management Portal. Select **[System] > [Configuration] > [Device Settings] > [SubTypes]**. When you click on “Edit” for each listed device sub-type, it displays a **Right Margin:** default option.

3.3.3.2 protocols

The 2nd positional parameter: A string of letter code characters enclosed in quotation marks (for example, "BNFU”), where each letter enables one of the terminal's rules for communicating. Letter codes are not case-sensitive. Letter codes may be specified in any order; because Caché executes them in left-to-right order, interactions between letter codes may dictate a preferred order in some cases. For a table of letter codes, see Letter Code Protocols.

A preceding plus or minus affects protocols as follows:

- No preceding plus or minus: New string replaces prior protocols string.
- Plus (+) precedes letter code string: Protocols in new string are added to prior protocols string.
- Minus (-) precedes letter code string: Protocols in new string are turned off, but other protocols remain in effect.

The + and – options for turning protocols on and off are not available in DSM-11 compatibility modes.

3.3.3.3 terminator

The 3rd positional parameter: A string of up to eight characters, any of which will terminate a **READ**. These terminators are in addition to those built into the protocols. See Using Terminators to End I/O Operations.

3.3.3.4 portstate

The 4th positional parameter: A string of up to eight bytes in positional order that govern the COM port state. The portstate bytes are as follows (bytes are numbered from 1 in left-to-right order):

Byte	Meaning	Values
1	Disconnect	D=disconnect (hangup) the port. blank=don't disconnect the port.
2	Modem Control	1=use modem control. 0=don't use modem control. blank=no change to modem control.

Byte	Meaning	Values
3	Data Bits	5=five data bits. 6=six data bits. 7=seven data bits. 8=eight data bits. blank=no change to data bit setting.
4	Parity	0=no parity. 1=odd parity. 2=even parity. 3=mark parity. 4=space parity. blank=no change to parity setting.
5	Stop Bits	1=one stop bit. 5=1.5 stop bits. 2=two stop bits. blank=no change to stop bit setting.
6	Flow Control	X=use Xon/Xoff flow control. C=use CTS/RTS flow control. D=use DSR/DTR flow control. N=disable flow control. blank=no change to flow control.
7	DTR Setting	0=disable DTR (set it off, keep it off). 1=enable DTR (set it on, keep it on). blank=no change to DTR state.
8	$ZA Error Reporting	0=disable $ZA error reporting (default). 1=enable $ZA error reporting. blank=no change to $ZA error reporting.

The following example shows a COM port state string:

ObjectScript

```
OPEN "COM2":(:::" 0801x0")
```

The string values are: blank (don't disconnect the port); 0 (don't use modem control); 8 (eight data bits); 0 (no parity); 1 (one stop bit); X (use Xon/Xoff flow control); 0 (disable DTR); default (disable $ZA error reporting).

The *Disconnect* parameter performs a hangup on modem-controlled ports by lowering the DTR signal for two seconds and then restoring it. A disconnect does not close the port; following a disconnect you can dial out again without reopening the COM device.

The *Modem Control* parameter determines how Caché responds to the state of the RLSD (Received Line Signal Detector) pin, also known as the DCD (Data Carrier Detect). If the line is modem controlled (modem control=1), Caché monitors the state of the RLSD, and generates an <ENDOFFILE> error if a **READ** command is issued when carrier is not present. Caché does not generate an error when a **WRITE** command is issued when carrier is not present. This is because it must be possible to send the dial command to the modem prior to a connection being established. Caché modem control can be enabled (1) or disabled (0) at any time. It is suggested that you disable modem control while sending commands to the modem, then enable modem control once carrier is detected and connection has been established.

The *DTR Setting* parameter is used to control login from an attached modem. If the DTR setting is 0 (zero), the DTR control signal is off, and modems cannot communicate with the computer. This prevents a dial-in connection from occurring. If the DTR setting is 1 (one), the DTR control signal is on, and modems can communicate with the computer. A dial-in connection can occur. If you configure DTR as off (0), then you must set it to on (1) with the **OPEN** command or **USE** command to be able to dial out using a connected modem. In most cases, the DTR setting is unimportant when using a null modem cable to connect directly to a terminal device or a serial printer. This is because the null modem cable should force the DTR control pin on.

The *$ZA Error Reporting* parameter enables reporting of the status of modem control pins to the **$ZA** special variable. This checking can be done regardless of the Modem Control byte setting for the COM port. If $ZA error reporting is enabled, COM port errors are cleared with a call to the Windows ClearCommError() function. The port error state is reported in the **$ZA** bits 16 through 22. For a table of **$ZA** bit values, refer to $ZA in the *Caché ObjectScript Reference*.

3.3.3.5 baud

The 5th positional parameter: an integer value that specifies the desired COM port baud rate. The following baud rates are supported: 110, 300, 600, 1200, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200, 128000, 256000.

3.3.4 Keyword Parameters for OPEN and USE Commands

The following table describes the keyword parameters for controlling terminal devices with both **OPEN** and **USE** commands. For each keyword, the table lists the corresponding Letter Code Protocols for **OPEN** and **USE**. Additional information on the use of these protocols can be found in the Letter Code Protocols table.

Table 3–3: OPEN and USE Keyword Parameters for Terminal Devices

Keyword	Default	Letter Code Protocols	Description
/BAUD=*n*			Corresponds to the *baud* positional parameter. /BAUD=*n* sets the modem baud rate for a COM port. Supported values are 110, 300, 600, 1200, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200, 128000, 256000.
/BREAK[=*n*] *or* /BRE[=*n*]	0	B	/BREAK or /BREAK=*n* for nonzero values of *n* enable the protocol. /BREAK=*n* for a zero value of *n* disables the protocol.
/COMPARAMS=*str*			Corresponds to the *portstate* positional parameter. (This keyword provides a way to specify a COM port state byte code string in a position-independent way.) The *portstate* byte codes that you can include in *str*, are listed in a table in earlier in this chapter.
/CRT[=*n*]	Depends on the operating system terminal setting	C and P	Associated with the C and P protocols. /CRT or /CRT=*n* for nonzero values of *n* enable the C protocol and disable the P protocol. /CRT=*n* for a zero value of *n* disables the C protocol and enables the P protocol.
/DISCONNECT			Corresponds to 1st byte of the *portstate* positional parameter. /DISCONNECT disconnects (hangs up) the COM port. It does not close the port; you can dial out again without reopening the COM device.
/ECHO[=*n*]	1	S	/ECHO or /ECHO=*n* for nonzero values of *n* disable the protocol. /ECHO=*n* for a zero value of *n* enables the protocol.
/EDIT[=*n*]	0	R and N	/EDIT or /EDIT=*n* for nonzero values of *n* enable the R protocol and disable the N protocol. /EDIT=*n* for a zero value of *n* disables the R protocol and enables the N protocol.

Keyword	Default	Letter Code Protocols	Description
/FLUSH[=*n*] *or* /FLU[=*n*]	0	F	/FLUSH or /FLUSH=*n* for nonzero values of *n* enable the protocol. /FLUSH=*n* for a zero value of *n* disables the protocol.
/GZIP[=*n*]	1		Specifies GZIP-compatible stream data compression. /GZIP or /GZIP=*n* (for nonzero values of *n*) enables compression on WRITE and decompression on READ. /GZIP=0 disables compression and decompression. Before issuing /GZIP=0 to disable compression and decompression, check the $ZEOS special variable to make sure that a stream data read is not in progress. /GZIP compression has no effect on I/O translation, such as translation established using /IOTABLE. This is because compression is applied after all other translation (except encryption) and decompression is applied before all other translation (except encryption).
/IMAGE[=*n*] *or* /IMA[=*n*]	0	I	/IMAGE or /IMAGE=*n* for nonzero values of *n* enable the protocol. /IMAGE=*n* for a zero value of *n* disables the protocol.
/IOTABLE[=*name*] *or* /IOT[=*name*]	If *name* is not specified, the default I/O translation table for the device is used.		Corresponds to the K\name\ protocol, which establishes an I/O translation table for the device.
/MARGIN=*n* *or* /MAR=*n*	0 (no margin)		Corresponds to the *margin* positional parameter, which sets the right margin for the terminal device.
/MODE=*n*	No default		Resets protocols and sets the terminal mode according to the value of *n*. *n*=0 sets LF and ESC as default terminators. *n*=1 is the same as mode 0 and enables the S protocol. *n*=2 is the same as mode 0 and enables the T protocol.

Keyword	Default	Letter Code Protocols	Description
/NOXY [=*n*]	0		No $X and $Y processing: /NOXY or /NOXY=*n* (for nonzero values of *n*) disables $X and $Y processing. This can substantially improve performance of READ and WRITE operations. The values of the $X and $Y variables are indeterminate, and margin processing (which depends on $X) is disabled. /NOXY=0 enables $X and $Y processing; this is the default.
/OBUFSIZE=*nnn*	256		Specifies the size of the terminal output buffer in bytes. Increasing the output buffer size can improve performance of screen painting with telnet over wide area networks with high latency. Valid values for /OBUFSIZE are 256 through 65536. The default is 256.
/PARAMS=*str* *or* /PAR=*str*	No default		Corresponds to the *protocols* positional parameter. (This keyword provides a way to specify a protocols letter code string in a position-independent way.) For a table of letter codes that you can include in *str*, see Letter Code Protocols.
/TERMINATOR=*str* *or* /TER=*str*	No default		Corresponds to the *terminator* positional parameter, which establishes user-defined terminators. To compose *str*, see Using Terminators to End I/O Operations.
/TPROTOCOL[=*n*] *or* /TPR[=*n*]	0	T	/TPROTOCOL or /TPROTOCOL=*n* for nonzero values of *n* enable the protocol. /TPROTOCOL=*n* for a zero value of *n* disables the protocol.
/TRANSLATE[=*n*] *or* /TRA[=*n*]	1	K	/TRANSLATE or /TRANSLATE=*n* for nonzero values of n enable I/O translation for the device. /TRANSLATE=*n* for a zero value of n disables I/O translation for the device.
/UPCASE[=*n*] *or* /UPC[=*n*]	0	U	/UPCASE or /UPCASE=*n* for nonzero values of n enable the protocol. /UPCASE=*n* for a zero value of *n* disables the protocol.

Keyword	Default	Letter Code Protocols	Description
/XYTABLE[=*name*] *or* /XYT[=*name*]	If *name* is not specified, the default $X/$Y action table for the device is used.	Y\name\	Corresponds to the Y\name\ protocol, which establishes a $X/$Y action table for the device.

3.3.5 Testing the Success of OPEN Commands

To determine whether an **OPEN** command succeeded, your code should test **$TEST** and/or **$ZE**. **$TEST** is only set if the **OPEN** command was specified with a *timeout* argument. A <NOTOPEN> error occurs only when **Ctrl-C** interrupts an **OPEN** command. Therefore, your code must not depend on <NOTOPEN> errors.

3.3.6 Letter Code Protocols for OPEN and USE

Special situations or terminals can require different protocols. With the *protocols* letter code parameter (or the corresponding keyword parameters) you can change the rules by which Caché communicates with the terminal. Protocols affect normal and single-character reads alike.

Normal mode, with all special protocols disabled, suffices for most terminal I/O. In normal mode Caché echoes each incoming ASCII character, sending it back to appear on the terminal. A **Return**, or a valid escape sequence, ends a **READ** command.

Issuing **OPEN** for a terminal turns off all previous protocols, except when you use the + and - options.

The following table describes valid *protocols* characters and their effects.

Table 3–4: Letter Code Protocols for OPEN and USE

Protocol Character	Name	Definition
B	BREAK	If breaks are enabled (+B), **Ctrl-C** interrupts a running routine with an <INTERRUPT> error. If breaks are disabled (-B), **Ctrl-C** does not cause an interrupt and "^C" is not displayed. The use of this protocol is dependent upon the **BREAK** command default established by the login mode, as follows: If you log in as programmer mode, interrupts are always enabled (**BREAK 1**). The B (or /BREAK) protocol specified in an **OPEN** or **USE** command has no effect. If you log in as application mode, **BREAK 0** is the default, and interrupts can be enabled or disabled by the B (or /BREAK) protocol specified in the **OPEN** or **USE** command.

Protocol Character	Name	Definition
C	CRT terminal	C mode accepts all eight bit characters as data, except for the following six: ASCII 3, 8, 10, 13, 21, and 127. The ASCII 127 Delete character echoes as a destructive backspace, that is, it backspaces and erases the preceding character. ASCII 21 (**Ctrl-U**) echoes enough destructive backspaces to bring the cursor to the start of the **READ**. If the setting for the right margin, or the nature of the terminal, forces echoed characters to begin a new line, **Ctrl-U** can erase only the characters on the last physical line. In any case, **Ctrl-U** cancels all input since the start of the **READ**. C is mutually exclusive with the P protocol.
F	Flush	Flush (empty) the input buffer before each **READ**. You can flush the input buffer to prohibit the user from typing ahead of **READ** operations on the terminal, because Caché discards anything typed between **READ** operations. Note that the command `WRITE *-1` flushes the input buffer at any time, regardless of the F protocol.
I	Image mode	I mode accepts all 256 eight bit characters as data, treating none as a READ terminator, except the termination character(s) (if any) that you explicitly specify in the *terminator* parameter. If you do not explicitly specify termination characters, you should use only single character and fixed length **READ** operations. Without defined termination characters, an ordinary **READ** results in a <TERMINATOR> error. Image mode (I) *protocol* can be combined with other *protocol* characters. In image mode, Caché ignores the protocol characters P, C and B. In image mode, the protocol characters F, N, R, S, and T remain in effect. When not in image mode, the device is in N (normal) mode.
K\name\ *or* Knum	I/O Translation Mode	When you use the K protocol for a device, I/O translation will occur for that device if translation has been enabled system-wide. You identify the previously defined table on which the translation is based by specifying the table's name. (The older form Knum, where "num" represents the number of the slot the table is loaded in, is being phased out but is still supported.)
N	Normal mode	N mode accepts all eight bit characters as data, except for the following six: ASCII 3, 8, 10, 13, 21, and 127. These implicit terminator and command line editing control characters, are described later in this chapter. If R (read line recall) protocol is enabled, N disables R protocol. This mode is the default if no *protocols* value is specified.
P	Print device	The ASCII Delete character echoes as a backslash (\), and **Ctrl-U** echoes as "^U" followed by a carriage return and line feed. When you issue an **OPEN** command for a terminal, Caché invokes the protocol C or P automatically, depending on the operating system terminal setting. These protocols remain in effect until you change the protocols for the device explicitly. A protocol string containing neither C nor P does not cancel this protocol.

Protocol Character	Name	Definition
R	Enable read line recall mode	The R protocol enables read line recall mode for that device. To activate read line recall for the current process, use the **LineRecall()** method of the %SYSTEM.Process class. To set the system-wide read line recall default, use the *LineRecall* property of the Config.Miscellaneous class. The R protocol overrides these default settings for the specified device. To change read line recall for an already-open device, you must explicitly specify another **OPEN** command to that device. Read line recall is disabled by specifying the N protocol.
S	Secret input	Nothing echoes on a **READ**. **READ** commands do not change **$X** and **$Y**. Read line recall is disabled.
T	Terminator	T mode does not treat any control characters as data. The following are control characters: ASCII characters with decimal values from 0 to 31 and 127 to 159. Most of these control characters are treated as READ terminator characters. The exceptions are the following control characters, which perform other control operations: ASCII 3 (**Ctrl-C**), ASCII 8 (backspace), ASCII 17 (**Ctrl-Q**), ASCII 19 (**Ctrl-S**), ASCII 21 (**Ctrl-U**), ASCII 24 (**Ctrl-X**), ASCII 27 (ESC), and ASCII 127 (DEL). When T mode is combined with I mode (IT protocol) all control characters (ASCII 0 to 31 and 127 to 159) are treated as READ terminator characters, with the exceptions of the output control characters **Ctrl-Q** (XOFF), and **Ctrl-S** (XON), and the input control characters **Ctrl-H** and **Ctrl-Y**. Output control characters **Ctrl-Q** and **Ctrl-S** are intercepted by most terminals and do not terminate a **READ** even in IT mode.
U	Upcase mode	U mode converts all input letters to upper case.
Y\name\ *or* Ynum	$X/$Y Action Mode	When you use the Y protocol for a device, the system uses the named $X/$Y Action Table. You identify a previously defined $X/$Y Action Table on used to translate by specifying the table name. If you don't know it, you can get the name from the system manager. $X/$Y action is always enabled. If Y is not specified and a system default $X/$Y is not defined, a built in $X/$Y action table is used. The + option works for turning on the Y protocol, but the - option does not. In order to disable a $X/$Y association, you can issue the command: `USE 0:(:"Y0")` (The older form Ynum, where *num* represents the number of the slot the table is loaded in, is being phased out but is still supported.)

3.3.6.1 Examples of Protocol Strings

The following series of examples show how protocol strings function. Each of the following USE commands builds on the protocol established by the preceding USE commands:

ObjectScript

```
USE 0:(80:"BP" )
```

The letter codes BP turn on the B and P protocols. This example enables breaks (B) and tells Caché to treat the terminal as a printing device (P).

ObjectScript

```
USE 0:(80:"P")
```

When it follows the **USE** command in the example just above, this command leaves the P protocol in effect, but turns off the B protocol.

ObjectScript

```
USE 0:(80:"+R" )
```

+R turns on read line recall, without affecting other protocol settings.

ObjectScript

```
USE 0:(80:"")
```

The empty string turns off all protocols. However, the P or C protocol remains in effect.

ObjectScript

```
USE 0:(80)
```

Omitting the protocol parameter leaves the protocol and explicit terminators unchanged.

3.3.7 Protocol Terminator Characters

OPEN and USE protocols define what READ input characters, control sequences, and keystrokes are treated as implicit terminator characters. These four protocols are I (image mode), N (normal mode (the default)), R (read line recall mode), and T (terminator mode):

- I (image mode) accepts all 256 eight bit characters as data, treating none as a READ input terminator or a command line editing character. Because of this, you should use only single character or fixed length **READ** operations in image mode. Without defined termination characters, an ordinary **READ** results in a <TERMINATOR> error.
- N (normal mode) and C (CRT mode) accept all characters as data except the following six: ASCII 3, 8, 10, 13, 21, and 127. Two of these, ASCII 10 (linefeed) and 13 (carriage return) terminate READ and submit input. ASCII 3 (**Ctrl-C**) discards input and issues an <INTERRUPT> error if BREAK is enabled. ASCII 8 (backspace) and 127 (delete) perform a single-character backspace erase then continue READ. ASCII 21 performs a multi-character backspace, erasing all prior characters, then continues READ.
- R (read line recall mode) accepts all characters as data except the following twenty: ASCII 1 through 8, 10 through 14, 16, 18, 21, 23, 24, 27, and 127. ASCII 10 (linefeed) and 13 (carriage return) terminate READ and submit input. ASCII 3 (**Ctrl-C**) discards input and issues an <INTERRUPT> if BREAK is enabled. The other characters perform the following command line editing functions:

  1 ^A = beginning of line
  2 ^B = back word
  3 ^C = interrupt
  4 ^D = delete current character
  5 ^E = end of line
  6 ^F = forward word
  7 ^G = delete to beginning of word ("wipe word backward")
  8 ^H = BS = destructive backspace
  9 ^I = HT = horizontal tab (echoed as a SPACE)
  10 ^J = LF = end of input
  11 ^K = VT = forward character
  12 ^L = FF = erase to end of line
  13 ^M = CR = end of input (same as LF)
  14 ^N = recall next input line
  16 ^P = recall previous input line
  18 ^R = back char (reverse)
  21 ^U = erase to start of line
  23 ^W = delete to end of word "gobble word forward")

24 ^X = erase entire line
27 ESC lead character for arrow and function keys
127 DEL = destructive backspace (same as BS)

- T (terminator mode) accepts all characters as data except the 65 control characters: ASCII 0 through 31 and ASCII 127 through 159. Most of these characters are treated as READ termination characters. This includes the tab character (ASCII 9), which is treated as a data character in all other protocols. A few characters are treated as command line control characters: ASCII 3 (**Ctrl-C**) discards input and issues an <INTERRUPT> if BREAK is enabled. ASCII 8 (backspace) and 127 (delete) perform a single-character backspace erase then continue READ. ASCII 21 (**Ctrl-U**) and ASCII 24 (**Ctrl-X**) perform a multi-character backspace, erasing all prior characters, then continues READ. ASCII 27 is the Escape character.
- IT (image mode + terminator mode) accepts all characters as data except the 65 control characters: ASCII 0 through 31 and ASCII 127 through 159. It treats all of the control characters as READ terminator characters.

In any of these modes you can explicitly specify additional terminator characters using the *terminator* parameter. Because image mode is commonly used for bit stream data, designation of any character as a terminator is usually avoided.

3.3.8 Explicit Terminator Characters

The *terminator* parameter in the **OPEN** or **USE** command lets you define specific characters as terminators for a **READ** or **WRITE** command. These explicit terminators can be used to supplement the terminator characters supplied by the specified *protocol*. The *terminator* parameter can also be used to override the designation of a character by the *protocol*, and instead designate it a terminator character. The exceptions to this ability to redefine a character as a terminator are: ASCII 0 (NULL), ASCII 3 (**Ctrl-C**), and the two output control characters **Ctrl-Q** (XON) and **Ctrl-S** (XOFF). These retain their functionality, and cannot be redefined as terminator characters.

3.3.8.1 Example

This example defines Z, **Backspace** and **Tab** as terminators for the principal device. The underscore is the concatenate operator.

ObjectScript

```
USE 0:(""::""):"Z"_$CHAR(8,9))
```

By issuing an **OPEN** command for an unowned terminal, you implicitly clear the Caché internal list of explicit terminators. When a protocol string appears, Caché then does the following:

1. Clears its list of explicit terminators.
2. Sets protocols according to the protocol string.
3. Copies a terminator string, if any, into the internal list of explicit terminators.

   The following table gives examples of explicit terminator strings.

Table 3–5: Terminator Strings: Examples

Terminator String	Definition
USE 0:(80:"C":$CHAR(27))	The Escape character terminates a **READ** rather than beginning an escape sequence.
USE 0:(80:"C":"")	The empty string clears all terminators.
USE 0:(80:"C")	Omitting the *terminator* parameter when you specify *protocol* clears all terminators.
USE 0:(80) or U 0:80	Omitting both *protocol* and *terminator* leaves terminators unchanged.

3.3.9 Summary of Protocols and Terminators in Read Operations

The following characters end a normal (nonimage) mode **READ**:

- **Return**
- Any character in the terminator string except ASCII NUL and the characters **Ctrl-C**, **Ctrl-O**, **Ctrl-Q**, and **Ctrl-S**.
- With the T protocol in effect, any control character except the output control characters. Control characters are non-printing characters with decimal values 0 to 31 and 127 to 159.
- Any escape sequence.
- Character number *y* of a fixed-length `READ x#y`.

The following characters end an image-mode **READ**:

- Any character specified in the terminator string except ASCII NUL.
- With the T protocol in effect, any control character.
- Character number *y* of a fixed-length `READ x#y`.

3.4 READ Command

Reads from 0 to 32 KB from the keyboard into the named variable. The *timeout* argument is optional. The command cannot end with a pound sign (#) or colon (:)

3.4.1 Syntax

```
READ variable:timeout          ; Variable-length read
READ variable#length:timeout   ; Fixed-length read
READ *variable:timeout         ; Single-character read
```

For more information, see READ in the *ObjectScript Language Reference*.

3.4.2 Examples

The following table gives several examples of how you use these arguments.

Table 3–6: READ Command Arguments: Examples

Example	Effect
READ ^GLO	Reads characters from the current device until it finds a terminator, and puts the resulting string in the global ^*GLO*.
READ X:60	Reads from the current device until it finds a terminator, and puts the string read into the variable *X*. Waits up to 60 seconds for the input to end before timing out. Striking a key does not reset the timeout value.
READ *X	Reads a single character from the current device and puts its decimal value in the local variable *X*.
READ X#1	Reads a single character from the current device and puts its string value into the local variable *X*.
READ X#45:60	Reads up to 45 characters from the current device and puts the string value into the local variable *X*. Waits up to 60 seconds for the input to end before timing out.

3.4.3 Read Line Recall

Read line recall mode provides line recall of editable lines as input for **READ** operations from a terminal. These recallable lines include both previous **READ** input lines and previous command lines. Echoing of input lines is a necessary precondition for read line recall.

Caché supports read line recall for both variable-length terminal reads (`READ variable`) and fixed-length terminal reads (`READ variable#length`). Caché does not support read line recall for single-character terminal reads (`READ *varaiable`). Read line recall supports the optional *timeout* argument.

For a fixed-length terminal read, the recalled line is truncated to one character less than the number of characters specified in the **READ**. This final **READ** character position is reserved for typing a line termination character, specifying an edit character, or adding one more data character.

When read line recall is active, you can provide input to a **READ** by using the **Up Arrow** and **Down Arrow** keys to recall a previous terminal input line. You can then use the **Left Arrow**, **Right Arrow**, **Home**, and **End** keys to position the cursor for editing the recalled line. You can use the **Backspace** key to delete a character, **Ctrl-X** to delete the entire line, or **Ctrl-U** to delete all of the line to the left of the cursor.

When read line recall is not active, the four **Arrow** keys, the **Home** key, and the **End** key all issue a line termination character. You can use the **Backspace** key to delete a single input character, and **Ctrl-X** (or **Ctrl-U**) to delete the entire input line.

You can use the **OPEN** or **USE** command to activate read line recall by specifying the R protocol, or to deactivate read line recall by specifying the N, I, S, or T protocol.

3.4.4 Special Protocol Characters Affect Terminal I/O

Each operating system intercepts certain protocol characters (UNIX®) or key combinations (such as **CTR-ALT-DEL** on Windows platforms), preventing these characters from affecting Caché. The console for Windows makes no attempt to override these operating system characteristics.

Other special characters can alter the way your routines execute, but do not appear in the **READ** command variable. Operating your terminal in image mode cancels these effects and causes Caché to treat these characters like any others.

READ is affected by output and input control characters. **READ** simply reads all other control characters, except termination characters. It does not echo them.

Output control characters affect both the flow and the output of a routine. These are described in the following table:

Table 3–7: Output Control Characters

Output Control Character	Decimal Value	Definition
Ctrl-C	3	If breaks are enabled, **Ctrl-C** interrupts a routine's execution. The routine behaves as though an <INTERRUPT> error has occurred. If breaks are disabled, **Ctrl-C** causes Caché to discard anything entered thus far in the current **READ**. You can use **Ctrl-C** to interrupt global module requests that require network operations. To trap **Ctrl-C**, set the special variable **$ZTRAP**. For additional information, see the section on enabling breaks.
Ctrl-S	19	**Ctrl-S** suspends output to the terminal. Output to the terminal resumes when Caché encounters a **Ctrl-Q**.
Ctrl-Q	17	**Ctrl-Q** resumes output suspended by **Ctrl-S**.

Input control characters affect input. Image mode (I protocol) treats these characters as data, but in normal mode they affect input to the current **READ** command. These characters are described in the following table:

Table 3–8: Input Control Characters

Input Control Character	Decimal Values	Definition
Delete	127	The Delete character removes the last character entered. If you press Delete repeatedly, you remove characters from right to left, but not beyond the beginning of the current **READ**. Delete uses a backspace to erase the last character on a CRT screen. Delete echoes as a backslash character ("\") on a printing terminal (such as a teletype).
Ctrl-U	21	Deletes either all characters you have entered since the start of the current **READ** or the contents of theUNIX® type-ahead buffer until the last carriage return. **Ctrl-U** erases the deleted characters on a CRT; on a printer it echoes ^U and sends a Return and LineFeed To flush the typeahead buffer completely, use **Ctrl-X**.
Ctrl-H	8	Performs the same function as Delete on some systems.
Return	13	A carriage return ends a **READ** in all protocols except "I" (image mode).
Escape	27	Begins an escape sequence. The sequence itself ends the **READ**, and **$ZB** contains the full sequence, including the leading Escape. Caché does not echo the characters of the sequence, but it does change **$X** and **$Y** unless you include the escape sequence in a `WRITE *` command. See $X and $Y and Cursor Position earlier in this chapter. An invalid escape sequence sets bit 8 of **$ZA**. Consider the example, `READ X`. After you enter the characters "AB", **Escape**, and "E", *X* will contain the two characters "AB", while **$ZB** contains the two characters Escape E. **$X** increases by two for the **AB**, but does not increase for the E.
LineFeed	10	Caché interprets LineFeed as a terminator for all terminal I/O.
Tab	9	Tab is a data value that echoes as a space, increases **$X** by one, and is stored as a Tab character in the string returned by the **READ**. This is true for all protocols except "T" (terminator). In "T" protocol, a tab is a terminator control character.

3.4.4.1 Disabling UNIX® Job Control

Using the UNIX® job control character, **Ctrl-Z**, within Caché can cause serious problems. For this reason, Caché disables **Ctrl-Z** automatically when you enter Caché on platforms whose UNIX® shell supports job control. Caché reenables **Ctrl-Z** when you exit Caché, and when you issue a **$ZF(-1)** call to execute a UNIX® shell command.

3.4.5 How the READ Command Processes Input

The **READ** command processes each character as it arrives from the input buffer. The following table shows how this processing occurs in normal mode. The figure below shows how the **READ** command processes image mode data.

Figure 3–1: READ Command Processing Normal (Non-Image) Mode

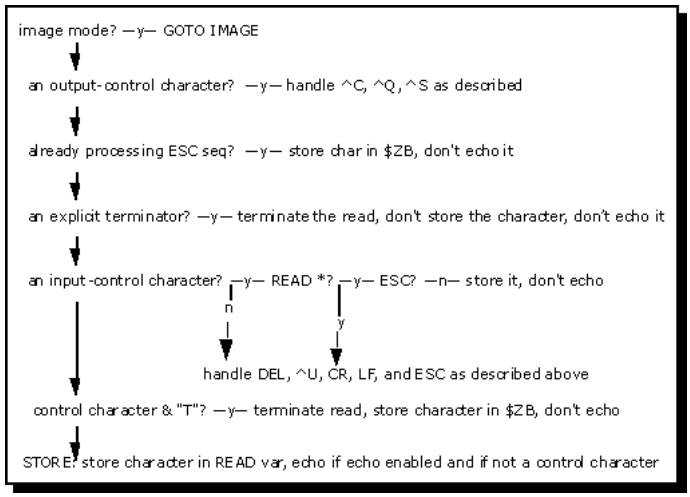
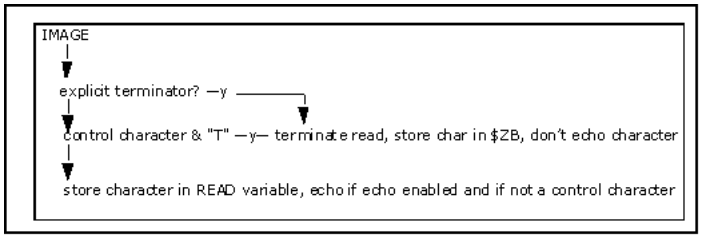


Figure 3–2: READ Command Processing Image Mode



3.5 WRITE Command

Writes zero or more characters to the terminal.

3.5.1 Syntax

```
WRITE *variable
WRITE *-n
WRITE #
WRITE /mnemonic
```

where

Argument	Definition
(none)	**WRITE** with no arguments writes all local variables on the current device.
*variable	`WRITE *variable` writes one character, whose decimal value equals *x*. The value of variable should be an integer in the range 0 to 255 for ASCII characters, and >255 for Unicode characters. By convention, values from 0 to 127 signify 7 bit ASCII characters, while 128 to 255, which represent the extended ASCII character set, relate to the application itself. If the hardware and software are properly set, Caché can handle 8 bit data. Example : You can use the eighth bit to represent international character sets. Caché routines often use `WRITE *` to send control characters for device dependent functions. Example : WRITE *27,*91,*50,*74 clears the terminal screen. `WRITE *` does not change **$X** or **$Y**; the assumption is that `WRITE *` output is highly specific to the output device.
*-1	`WRITE *-1` clears the input buffer when the next **READ** is issued. It clears any characters that are pending for the next **READ** command. Thus all type-ahead characters are cleared. An input buffer holds characters as they arrive from the keyboard, even those you type before the routine executes a **READ** command. Thus you can answer questions even before they appear on the screen. When the **READ** command takes characters from the buffer, Caché echoes them back to the terminal so that questions and answers appear together. When a routine detects errors, it may issue `WRITE *-1` to cancel these answers.
*-10	`WRITE *-10` clears the input buffer immediately. It does not wait for the next **READ** command. Thus it clears all type-ahead characters issued before the `WRITE *-10`; type-ahead characters issed after the `WRITE *-10` remain in the input buffer for use by the next **READ**.
#	Issuing `WRITE #` to a CRT terminal clears the screen and sends the cursor to the home (0,0) position. For a hardcopy terminal, it writes a Carriage Return and Form Feed. `WRITE #` sets **$Y** to 0.
/mnemonic	Issuing `WRITE /mnemonic` causes Caché to interpret mnemonic as defined in the active mnemonic space. If there is no active mnemonic space, an error results. You can specify the active mnemonic space in two ways: By naming a default mnemonic space for each device type using the Namespace and Network Configuration editor by specifying a mnemonic space in the **OPEN** or **USE** command for the device. For more information, see Controlling Devices with Mnemonic Spaces.

For more information, see WRITE in the *ObjectScript Language Reference*.

3.5.2 Examples

In the following example, `WRITE *` rings the bell on the user's terminal, displays a prompt, and clears the input buffer of any characters received but not yet used.

ObjectScript

```
SET eek="No. I can't accept that reply"
WRITE *7,eek,*-10
```

The following two examples show the difference between `WRITE *-1` and `WRITE *-10`. In both cases, the user responds to the first **READ** and presses ENTER, then types ahead during the two pauses caused by the **HANG** commands:

ObjectScript

```
READ "type:",x HANG 4 WRITE *-1 HANG 4 READ "type:",y
```

In the above example, Caché clears the input buffer when the second **READ** is issued. Thus any text typed during either **HANG** is cleared from the buffer.

ObjectScript

```
READ "type:",x HANG 4 WRITE *-10 HANG 4 READ "type:",y
```

In the above example, Caché immediately clears the input buffer when **WRITE *-10** is issued. Thus any text typed during the first **HANG** is cleared, but any text typed during the second **HANG** is supplied to the second **READ** command.

In the following example, `WRITE /mnemonic` uses the control mnemonic CUP (CUrsor Position) to move the cursor to the third column of the fourth line on the terminal. In this example, the predefined mnemonic space ^%X364 is specified in the **USE** command, and the name of an open terminal device is specified using the *terminal* variable. See Predefined Mnemonic Spaces for Terminals for a description of ^%X364.

ObjectScript

```
USE terminal:(80:"BP"):"%X364"
SET %1=3,%2=4
WRITE /CUP(%1,%2)
```

3.6 CLOSE Command

Releases ownership of the device, which is gained with an **OPEN** command.

3.6.1 Syntax

```
CLOSE device
```

For more information, see CLOSE in the *ObjectScript Language Reference*.

3.7 Predefined Mnemonic Spaces for Terminals

Caché provides two predefined mnemonic spaces for use with terminals:

- ^%X364 for ANSI X3.64 terminals
- ^%XDTM for DTM PC Console

If you make one of these mnemonic spaces active, you can use the control mnemonics associated with them in `WRITE /mnemonic` commands. You can also create your own mnemonic spaces. See Controlling Devices with Mnemonic Spaces in the chapter Terminal I/O for more information on mnemonic spaces.

The following sections describe the control mnemonics for these mnemonic spaces.

3.7.1 Mnemonic Space for X3.64

Caché provides a built-in mnemonic space for the ANSI X3.64 definition. This mnemonic space is the Caché routine %X364 in the manager's namespace. To use routine %X364, either:

- Have your Caché system manager enter %X364 as the default mnemonic space in the **IO Settings** configuration setting. From the Management Portal, select **[System] > [Configuration] > [Device Settings] > [IO Settings]**.
- Issue an **OPEN** command specifying this mnemonic space:

ObjectScript

```
OPEN "terminal"::"^%X364"
```

The following table lists the mnemonics.

Table 3–9: Control Mnemonics for %X364 Mnemonic Space

Calling Sequence	Name	System Variable Affected
APC	Application Program Command	
BEL	Ring the bell	
CBT(%1)	Cursor Backward Tabulation	**$X**
CCH	Cancel Character	
CHA(%1)	Cursor Horizontal Absolute	**$X**
CHT(%1)	Cursor Horizontal Tabulation	**$X**
CNL(%1)	Cursor Next Line	**$X**,**$Y**
CPL(%1)	Cursor Preceding Line	**$X**,**$Y**
CPR	Cursor Position Report	
CTC(%1,%2,%3,%4, %5,%6,%7,%8,%9)	Cursor Tabulation Control	
CUB(%1)	Cursor Backward	**$X**
CUD(%1)	Cursor Down	**$Y**
CUF(%1)	Cursor Forward	**$X**
CUP(%1,%2)	Cursor Position	**$X**, **$Y**
CUU(%1)	Cursor Up	**$Y**
CVT(%1)	Cursor Vertical Tabulation	**$Y**
DA	Device Attributes	
DAQ(%1,%2,%3,%4, %5,%6,%7,%8,%9)	Define Area Qualification	
DCH(%1)	Delete Characters	
DCS	Device Control String	

Calling Sequence	Name	System Variable Affected
DL(%1)	Delete Lines	
DMI	Disable Manual Input	
DSR(%1)	Device Status Report	
EA(%1)	Erase in Area	
ECH(%1)	Erase Characters	
ED(%1)	Erase in Display	
EF(%1)	Erase in Field	
EL(%1)	Erase in Line	
EMI	Enable Manual Input	
EPA	End of Protected Area	
ESA	End of Selected Area	
FNT	Font Selection	
GSM	Graphic Size Modification	
GSS	Graphic Size Selection	
HPA(%1)	Horizontal Position Attribute	**$X**
HPR(%1)	Horizontal Position Relative	**$X**
HTJ	Horizontal Tab with Justify	**$X**
HTS	Horizontal Tab Set	**$X**
HVP(%1,%2)	Horizontal and vertical position	**$X**, **$Y**
ICH(%1)	Insert Characters	
IL(%1)	Insert Lines	
IND	Index	**$Y**
INT	Interrupt	
JFY	Justify	
MC	Media Copy	
MW	Message Waiting	
NEL	Next Line	**$X**, **$Y**
NP(%1)	Next Page	
OSC	Operating System Command	
PLD	Partial Line Down	**$Y**
PLU	Partial Line Up	**$Y**
PM	Privacy Message	

Calling Sequence	Name	System Variable Affected
PP(%1)	Preceding Page	
PU1	Private Use 1	
PU2	Private Use 2	
QUAD	QUAD	
REP(%1)	REPEAT	**$X**, **$Y**
RI	Reverse Index	**$Y**
RIS	Reset to Initial State	**$X**=0 **$Y**=0
RM(%1,%2,%3,%4,%5,%6,%7,%8,%9)	Reset Mode	
SEM	Select Editing Extent Mode	
SGR(%1,%2,%3,%4,%5,%6,%7,%8,%9)	Select Graphic Rendition	
SL	Scroll Left	
SM(%1,%2,%3,%4,%5,%6,%7,%8,%9)	Set Mode	
SPA	Start of Protected Area	
SPI	Spacing Increment	
SR	Scroll Right	
SS2	Single Shift Two	
SS3	Single Shift Three	
SSA	Start of Selected Area	
ST	String Terminator	
STS	Set Transmit State	
SU	Scroll Up	
TBC	Tabulation Clear	
TSS	Thin Space Specification	
VPA(%1)	Vertical Position Attribute	**$Y**
VPR(%1)	Vertical Position Relative	**$Y**
VTS	Vertical Tab Set	

3.7.2 Mnemonic Space for DTM PC Console

Caché provides the Caché routine %XDTM to match the mnemonics used in developing applications for DTM. This mnemonic space is available but is not set up as the default mnemonic space for terminals. If you port applications created for DTM to Caché, you can either:

- Configure ^%XDTM as the default mnemonic space for terminals (**MnemonicTerminal**) in the Management Portal, or
- Reference the ^%XDTM mnemonic space in the **OPEN** or **USE** command.

3.7.3 DTM Examples

3.7.3.1 UNIX®

ObjectScript

```
OPEN "/dev/tty04/"::"^%XDTM"
```

3.7.3.2 Windows

ObjectScript

```
OPEN "c:\sys\user"::"^%XDTM"
```

Then Caché can correctly interpret the DTM control mnemonics in `WRITE /mnemonic` commands, shown in the following table.

Table 3–10: Control Mnemonics for DTM PC Console

Mnemonic	Description
AA	Normal mode
AB	Bold mode
AC	Underlined mode
AD	Bold, underlined mode
AE	Reverse video
AF	Reverse video/Bold mode
AG	Reverse video/Underline mode
AH	Reverse video/Bold, underlined mode
AI	Blink mode
AJ	Bold, blink mode
AK	Underlined, blink mode
AL	Bold, underlined, blink mode
AM	Reverse video / Bold, blink mode
AN	Reverse video / Bold, blink mode
AO	Reverse video / Underlined, blink modes
AP	Reverse video / Bold, underlined, blink modes
AZ	Mode Z
B(%1,%2)	Set video attributes: %1 provides attribute for characters, %2 provides attribute for clearing frames
BOX	Draw a window-relative utility box

Mnemonic	Description
C(%1,%2)	Position cursor at column %1, line %2
CLR	Clear current frame
COLOR(%1,%2)	Set IBM PC Color: Foreground %1, Background %2
DC(%1)	Delete %1 characters
EC(%1)	Erase %1 characters
EF	Erase to end of frame
EL	Erase to end of line
F(%1,%2,%3, %4,%5)	Fill rectangular area with $CHAR(%1) at upper left corner, %4 columns wide by %5 lines high
GETCUR	Return terminal cursor position
HF	Screen half bright off
HIDECURSOR	Hide mouse cursor
HN	Screen half bright
IC(%1)	Insert %1 characters
LF	Disable literal mode
LN	Enable literal mode, which displays control characters graphically on a PC screen.
MARK(%1)	Make mark on screen
NORM	Enable normal display attributes
PAD(%1)	Write %1 NULLS for padding
PF	Pause off
PN	Pause on
RF	Screen reverse video off
RN	Screen reverse video
SD(%1,%2,%3)	Scroll current frame down by %3 lines
SHOWCURSOR	Show mouse cursor
SU(%1,%2,%3)	Scroll current frame up by %3 lines, starting at line %1 down to but not including line %2
VF	Visible cursor off
VN	Visible cursor on
WBOX	Draw a screen-relative utility box
WCLOSE	Close utility window
WINDOW	Set scrolling window
WOPEN	Open utility window
Y(%1)	Set binary frame attribute

3.8 PRINT and ZPRINT Commands

Writes one or more lines of the currently loaded Caché routine to the current device.

ZPRINT has the same effect and arguments as **PRINT**.

3.8.1 Syntax

```
PRINT
ZPRINT
PRINT x
ZPRINT x
PRINT x:y
ZPRINT x:y
```

where

Argument	Definition
(none)	The **PRINT** or **ZPRINT** command with no arguments prints the entire routine.
x,*y*	The variables *x* and *y* indicate the range of lines to print. They can be either a line reference of the form TAG+OFFSET, or a line number of the form +7. Referring to a line not in the routine implies the empty line following the routine's last line. *x* = First or only line to print. *y* = Last line to print.

For more information, see PRINT in the *ObjectScript Language Reference*.

3.8.1.1 Example

This example prints the first line of the current routine, four lines starting at INIT, and all the lines from FINI to the end:

ObjectScript

```
INIT
  SET a=1
  SET b=2
  SET c=3
  SET d=4
FINI
  SET x=24
  SET y=25
  SET z=26
  PRINT +1,INIT:INIT+3,FINI:+9999
```

3.9 Programming Your Terminal

3.9.1 Using Caché to Program Formatted CRT Screens

Several features of Terminal I/O aid in programming formatted screens:

- Use `WRITE *` to send control sequences easily.
- Use **READ** to receive escape-sequence responses.
- Use `SET $X = expression` and `SET $Y = expression` to update the current cursor position.

Fixed-length **READ** and programmer-specified termination characters make it convenient to read individual fields. You can use the Secret protocol to make passwords invisible.

Remember that `WRITE *` does not change **$X** or **$Y**. If you want to change them, use `WRITE $C(X)`, or simply set them explicitly.

3.9.1.1 Example

This example sets the VT100 cursor to line 10, column 20

```
%SYS>SET DY=10,DX=20
%SYS>WRITE *27,*91,DY+1,*59,DX+1,*72 SET $Y=DY,$X=DX
```

3.9.1.2 Use CURRENT^%IS to Set Variables

The utility routine CURRENT^%IS sets some useful local variables to work for the current device. To call this routine, enter:

```
%SYS>DO CURRENT^%IS
```

This command sets the variables indicated in the following table.

Table 3–11: Features Enabled By CURRENT^%IS

Code	Definition
W @FF	Clears the screen and moves the cursor to the upper left corner (column 0, line 0) leaving **$X**=0, **$Y**=0.
S DX=42,DY=10 X XY	Moves the cursor directly to column 42, line 10, leaving **$X**=42, **$Y**=10.

3.9.2 Programming Escape Sequences

The ANSI standard for escape sequences makes programming of smart terminals practical. The **Escape** character and all characters after it in a string do not display on the screen, but do update **$X** and **$Y**. Send escape sequences to the terminal with `WRITE *` statements and keep **$X** and **$Y** up to date by setting them directly.

The ANSI standard establishes a standard syntax for escape sequences. The effect of a particular escape sequence depends on the type of terminal you are using.

Look for incoming escape sequences in **$ZB** after each **READ**. Caché puts ANSI-standard escape sequences and any others that use the ANSI forms in **$ZB**. Caché recognizes two forms of escape sequence:

3.9.2.1 Regular form

- An **ESC**.
- Optionally the character “O” (the letter), decimal value 79.
- Zero or more characters with decimal values 32–47.
- One character with decimal value 48–126.

3.9.2.2 Control form

- The **ESC** character, decimal value 27.
- The “[” character, decimal value 91.

- Zero or more characters with decimal values 48–63.
- Zero or more characters with decimal values 32–47.
- One character with decimal value 64–126.

Furthermore, the sequence can be no longer than 16 characters. Escape sequences that violate these forms or rules set bit 8 of **$ZA**, whose value is 256.

3.9.3 Example

Assume that you are programming a terminal whose **Help** key sends the two-character sequence **Escape-?** (**?** has a decimal value of 63)

```
%SYS>SET HELP=$C(27,63)
ASK READ !,"Enter ID: ",X I $ZB=HELP Do GIVEHELP GoTo ASK
```

Your routine can detect nonstandard escape sequences as follows:

1. Make **ESC** a terminator.
2. When **ESC** appears in **$ZB**:
   a. Disable echo with the Secret protocol to prevent modification of $X/$Y.
   b. Read the rest of the sequence with `READ *`;
   c. Turn off Secret to re-enable echo.

   In the following figure, the user is asked to enter an ID. If the user presses **Esc-?**, a Help screen appears. The subroutine **ESCSEQ** assumes that nonstandard escape sequences end with an asterisk “*”.

ObjectScript

```
DEMOS
  SET HELP=$C(27,63) ;Get Help with <ESC>?
    SET ESC=$C(27) USE 0:(""":""":ESC) ; Make <ESC> a READ terminator
                                       ; character
ASK READ !,"Enter ID: ",X I $ZB=ESC Do ESCSEQ G:SEQ=HELP ASK
    . ;Input ID. Handle Help request.
    .
    Quit
HELPSCR  ;Process Help request
    .
    Quit
ESCSEQ  USE 0:(""":"S") SET SEQ=ESC ;Set terminal to no echo,init SEQ
    FOR I=1:1 {
      READ *Y
      SET SEQ=SEQ_$C(Y)
      QUIT:Y=42 }
    ; Read in Escape sequence,
    ; end at "*"
    USE 0:(""":""":ESC) Quit ;Redefine terminator
```

3.9.4 Caché Supports Full or Half Duplex and Echo

Caché prefers that you use full duplex terminals; in other words, your keyboard should operate independently from your printer or screen.

Full duplex means simultaneous and independent transmission in both directions. Half duplex means transmission in only one direction at a time. Duplex has nothing to do with echo, although you may see a terminal marked full duplex for remote echo and half duplex for local echo. This designation means that the terminal displays the characters you type and does not expect Caché to echo them.

Set your terminal to local echo off or full duplex, letting Caché provide the echo. The echo comes not when the computer receives the character, but when the **READ** command takes it from the input buffer; therefore, the prompts and answers of a dialog keep their intended positions on the screen regardless of whether the user types ahead.

Some public networks provide their own echo to the terminal.

On Windows systems, consoles do not permit local echo setup changes. For terminals attached via a terminal emulator (e.g., VT220), refer to your terminal emulator documentation for instructions to disable local echo.

On UNIX® systems, use the **stty** command to avoid double echoes while keeping **$X** and **$Y** in agreement with the cursor's position.

3.9.5 Caché Supports Intercomputer Links and Special Devices

Caché provides flexible protocols and large independent buffers enable routines to deal with unusual devices and their protocols. For example, Caché easily supports full duplex communication between two computers on a terminal-to-terminal link. Two Caché systems require only a physical connection, the right protocols, and identical settings of speed, parity, and character length. With the aid of external converters, Caché communicates with IBM ports as a synchronous EBCDIC terminal.

Keep these points in mind when designing an intercomputer link:

- Turn off echo at both ends by including the S protocol in **OPEN** or **USE**, or by using the operating system's terminal parameters.
- Unless your communication protocol supports XON/XOFF flow control (**Ctrl-Q** and **Ctrl-S**), be sure it limits unacknowledged transmissions to the limit of the operating system's input buffering; otherwise you may lose data.
- In image mode, Caché does not support XON/XOFF. In nonimage (normal) mode, the operating system's terminal parameters determine whether the computer issues an XOFF if the operating system's input buffer is almost full. If XOFF and XON are not supported, make the buffer large enough that you do not need them.
- Test **$ZA** after read operations to detect transmission errors such as parity or data overrun conditions.

4

Local Interprocess Communication

This chapter describes how to set up communication between local Caché processes, and with other processes outside of Caché.

- Using Pipes to Communicate with Processes
- Communication Between Caché Processes

For information on remote Client/Server communications using TCP/IP, refer to the TCP Client/Server Communication chapter of this manual.

4.1 Using Pipes to Communicate with Processes

You can communicate between your Caché processes and external UNIX® or Windows processes through a pipe, just as at the UNIX® or Windows operating system level. You can send output to or receive input from the pipe. The pipe is one-way; you cannot read from and write to the same program at the same time.

When you open a pipe to another program for output, you can write to it as if it were a sequential file. The program then uses what you have written as its input stream. This capability is especially helpful when you want Caché processes to share resources with external processes.

For example, many users run Caché and a word processing program together and want the two applications to share printers properly. Caché assumes it has full access and responsibility for letting processes send information to devices. However, most UNIX® applications rely on a standard UNIX® utility, **lpsched**, to schedule access to the printer and spooling files.

When these UNIX® applications need to print, they call a utility called **lp** or **lpr**, instead of writing directly to the printer port. The **lp** (or **lpr**) utility then invokes **lpsched**, which in turn schedules access to the printer for the job from which **lp** (or **lpr**) was called. When you use **lp**, you do not need to wait for printing to occur. As soon as you have finished writing your print job to **lp**, you simply close the file; **lp** takes care of spooling the job to disk while awaiting its turn to print.

Caché lets you join in this cooperative environment by an extension to the **OPEN** command. You can issue this command directly, or through Caché utilities that use it.

4.1.1 Opening Pipes to Caché Utilities

You can open a pipe to a Caché utility as well as to UNIX® or Windows processes. Before you can use a pipe for utility I/O, your system manager must define the pipe device on your Caché system.

After the system manager defines the pipe device, when you run a utility (such as **%RD**), you answer the "Device:" prompt with the mnemonic the system manager defined. Your output goes automatically to that device.

4.1.2 Pipes and Command Pipes

Caché supports both standard pipes and command pipes. Standard pipes are used for relative short command strings, in which the command name and its arguments are less than 256 characters. Command pipes are used when the command string is 256 characters or more in length. In both cases, pipes can only be used on UNIX® and Windows systems.

4.1.2.1 Standard Pipe OPEN

The following is the **OPEN** command syntax for standard pipes:

```
OPEN program:(parameters):timeout
```

Because *program* is the first argument (the device argument), it must follow the **OPEN** command device name limitation of 256 characters.

4.1.2.2 Command Pipe OPEN

The following is the **OPEN** command syntax for command pipes:

```
OPEN cpipename:program:timeout
OPEN cpipename:(program:parameters:::closetimeout):timeout
```

The *cpipename* argument can take the value `"|CPIPE|"` if there is only command pipe open concurrently. To open multiple concurrent pipes, specify `"|CPIPE|xxxxxx"`, where *xxxxxx* represents a user-specified unique identifier. This *cpipename* argument is the argument specified for subsequent **USE** and **CLOSE** commands.

Because *program* is the second argument, it is not limited to 256 characters. The maximum length of *program* is platform dependent.

4.1.3 OPEN Command for Interprocess Communication

The **OPEN** command allows your program to communicate with processes external to Caché.

4.1.3.1 OPEN Arguments

Argument	Description
cpipename	*Command Pipes Only* — either `"|CPIPE|"` or `"|CPIPE|xxxxxx"`, where *xxxxxx* represents a user-specified unique identifier.

Argument	Description

program	A command pipe can execute a *program* with a command shell, or without a command shell (directly). Executing without a command shell is preferred in most situations. A standard pipe executes a *program* with a command shell.
	
	Command Pipes Only — To execute without a command shell, specify /COMMAND=*program*. If *program* has arguments, you must specify them using the /ARGS keyword. If you specify either the /COMMAND or /ARGS keyword, the *program* is executed without a command shell: `(/COMMAND=program)`, `(/COMMAND=program:/ARGS=arg1)` and `(program:/ARGS=arg1)` are all valid syntax. /ARGS can take a single argument, a comma-separated list of arguments, or an array. For example, `(/COMMAND=program:/ARGS=arg1,arg2)`. You can specify a variable number of arguments using an array:
	
	ObjectScript
	
	`SET array(1)=arg1, array(2)=arg2, array=2`
	`OPEN device:(/COMMAND=cmd:/ARGS=array...)`
	
	To execute using a command shell, specify *program*, omitting both the /COMMAND and /ARGS keywords.
	
	The *program* string contains the full pathname of a program installed on your system. It contains the command name and its arguments (if any) to be executed on the host system. For a standard pipe, limited to <256 characters. For command pipe, maximum length is platform dependent, but substantially more than 256 characters.

parameters	Read. For a standard pipe specify "Q" or "QR" to open a queue or pipe to accept input from another process. For a command pipe: because a command pipe is unambiguously a pipe, the "Q" letter code is not required; specify "R". Write. For a standard pipe specify "QW" to open a queue to send input to another process. For a command pipe: because a command pipe is unambiguously a pipe, the "Q" letter code is not required; specify "W". You can specify these and other parameters using the */keyword* parameters, separated by colons. For example, `OPEN "|CPIPE|":(cmd:/READ:/IOTABLE="UTF8")`. The following optional keyword parameters are commonly used with pipes: "K/*name*/" (or "K*num*") to enable I/O translation, if translation has been enabled system-wide. You identify the previously defined table on which the translation is based by specifying the table's name. The "+" and "-" options for turning protocols on and off are not available with the "K" protocol. "Y/*name*/" (or "Y*num*") to tell the system to use the named $X/$Y Action Table. You identify the previously defined $X/$Y Action Table on which translation is based by specifying the table's name. $X/$Y action is always enabled. If Y is not specified and a system default $X/$Y is not defined, a built in $X/$Y action table is used. The "+" and "-" options for turning protocols on and off are not available with the Y protocol. You can specify the "S" (stream), "F" (fixed length), or "U" (undefined length) mode parameters with the above parameters. You *cannot* specify the "V" (variable length) mode parameter. For a complete list of letter code and keyword parameters, refer to OPEN Mode Parameters in the "Sequential File I/O" chapter of this manual.

Argument	Description
closetimeout	*Optional* — UNIX® only: You can specify the number of seconds the **CLOSE** command will wait for the command process to exit when closing a piped command device. The default is 30 seconds. You can override this *closetimeout* by specifying an "I" (immediate) argument on the CLOSE command for interprocess communication.
timeout	*Optional* — A positive integer whose value in seconds is the longest time Caché waits for an **OPEN** to successfully finish. If Caché is able to open interprocess communication before the timeout expires, it sets **$TEST** to 1. If Caché is not able to open interprocess communication before the timeout expires, it sets **$TEST** to 0. If you omit the timeout or specify 0, the **OPEN** returns control to the process immediately.

4.1.3.2 OPEN Command Pipe Examples

The following are valid command pipe **OPEN** statements. Each example specifies a timeout of 10 seconds:

ObjectScript

```
OPEN "|CPIPE|1":"/nethome/myprog":10                                    // using shell, no args
OPEN "|CPIPE|1":("/nethome/myprog":/WRITE):10                          // using shell, no args, WRITE

OPEN "|CPIPE|2":/COMMAND="/nethome/myprog":10                           // no shell, no args
OPEN "|CPIPE|3":("":/COMMAND="/nethome/myprog"):10                      // no shell, no args
OPEN "|CPIPE|4":(/COMMAND="/nethome/myprog":/ARGS=arg1):10              // no shell, 1 arg
OPEN "|CPIPE|5":("/nethome/myprog":/ARGS=arg1):10                       // no shell, 1 arg
OPEN "|CPIPE|6":("/nethome/myprog":/ARGS=arg1:/WRITE):10                // no shell, 1 arg, WRITE
OPEN "|CPIPE|7":(/COMMAND="/nethome/myprog":/ARGS=arg1,arg2):10         // no shell, 2 args
OPEN "|CPIPE|8":(/COMMAND="/nethome/myprog":/ARGS=args...:/WRITE):10  // no shell, args array, WRITE
```

On a Windows system, an argument can include a blank space or a double quote (") character. In these cases, the argument can be quoted, and a literal double quote character can be escaped by doubling it:

ObjectScript

```
OPEN "|CPIPE|9":("/nethome/myprog":/ARGS="string with blanks"):10
OPEN "|CPIPE|10":("/nethome/myprog":/ARGS="string with literal "" character"):10
```

4.1.3.3 Errors

If you issue an **OPEN** command with the "QW" parameter for a non-IPC device, a <WRITE> error occurs when you try to write to this device.

The following UNIX® example opens an output pipe to the lp program, whose pathname in this case is /usr/bin/lp. Then it sends output from the global ^*TEXT* to the printer through this pipe.

ObjectScript

```
print ; Send the first layer of global ^TEXT to the printer.
      SET IO="/usr/bin/lp"
      OPEN IO:"QW"   ; Open the pipe to lp
      USE IO WRITE "The first layer of ^TEXT",!   ; Print the title
      ;  . . .
      ;  Print each line, using $ORDER on the global ^TEXT
      USE IO WRITE !,"The End.",#
      CLOSE IO    ; close the pipe, spooling the file to lpsched
      QUIT
```

You can alter this example so that the **OPEN** command passes arguments to the lp program. For example, to specify that **lp** should send the output to the printer device named laserjet, you could replace the **SET** command with the following:

ObjectScript

```
   SET IO="/usr/bin/lp -dlaserjet"
```

The following example shows how to read from an external program. Here the process opens an input pipe to the UNIX® program who, so that it can read the IDs of all users who are currently logged in to UNIX®.

ObjectScript

```
getids ; read the login IDs of everybody currently on
       SET IO="/usr/bin/who"
       SET $ZTRAP="EOT"
       KILL LOGINS
       OPEN IO:"Q"
       ; note that "R" (the default) is understood
       SET users=0
       FOR I=0:0 {
          USE IO
          READ USER
          SET users=users+1
          SET LOGINS(USER)=""
          }
       QUIT
EOT    SET $ZTRAP=""
       USE 0
       WRITE !,USERS," is/are currently logged on.",!
       CLOSE IO
       QUIT
```

On a Windows system, when a CPIPE **OPEN** *program* argument specifies /COMMAND or /ARGS, the system uses CreateProcess() to run the command. If the CreateProcess() fails, the **OPEN** will fail with a <NOTOPEN> error. The GetLastError() value is available via **$SYSTEM.Process.OSError()**.

On a UNIX® system, when a CPIPE **OPEN** *program* argument specifies /COMMAND or /ARGS, the system creates a new process which issues an exec() to run the command. If the exec() fails, the **OPEN** will fail with a <NOTOPEN> error. The exec() errno is available via **$SYSTEM.Process.OSError()**.

4.1.3.4 OPEN and USE Command Keywords

The following table describes the keywords for controlling interprocess communications pipes with both **OPEN** and **USE** commands.

Table 4–1: OPEN and USE Command Keywords for Interprocess Communications Pipes

Keyword	Default	Description
/IOTABLE[=*name*] *or* /IOT[=*name*]	If *name* is not specified, the default I/O translation table for the device is used.	Corresponds to the K\name\ parameter code, which establishes an I/O translation table for the device.
/TRANSLATE[=*n*] *or* /TRA[=*n*]	1	Associated with the K parameter code. /TRANSLATE or /TRANSLATE=*n* for nonzero values of *n* enable I/O translation for the device. /TRANSLATE=*n* for a zero value of *n* disables I/O translation for the device.
/XYTABLE[=*name*] *or* /XYT[=*name*]	If *name* is not specified, the default $X/$Y action table for the device is used.	Corresponds to the Y\name\ parameter code, which establishes a $X/$Y action table for the device.

4.1.3.5 OPEN-only Keywords

The following table describes the keywords for controlling interprocess communications pipes with only the **OPEN** command.

Table 4–2: OPEN-only Command Keywords for Interprocess Communications Pipes

Keyword	Default	Description
/IGNOREEOF[=*n*] *or* /IGN[=*n*]	0	Corresponds to the I parameter code, which specifies that a **READ** operation should be retried (ignoring any EOF condition) indefinitely or until the specified timeout expires. /IGNOREEOF or /IGNOREEOF=*n* for nonzero values of *n* enable the parameter code and /IGNOREEOF=*n* for a zero value of *n* disables the parameter code.
/PARAMS=*str* *or* /PAR=*str*	No default	Corresponds to the parameter codes positional parameter. (It provides a way to specify a parameter code string in a position independent way.)
/QUEUE *or* /QUE	The device is not recognized as an interprocess communications pipe.	Corresponds to the "Q" parameter code, which specifies that an interprocess communications pipe should be opened. Note that using this command requires Use permission on the %System_Callout resource.
/Read	Read is the default if neither /Read nor /Write is specified.	Corresponds to the "R" parameter code, which specifies that a queue or pipe should be opened to accept data from another process.
/Write *or* /WRI	Read is the default if neither /Read nor /Write is specified.	Corresponds to the "W" parameter code, which specifies that a queue or pipe should be opened to send data to another process.

4.1.4 READ Command for Interprocess Communication

4.1.4.1 Syntax

```
READ:pc readargument,...
```

READ reads data from a pipe.

where *readargument* can be:

```
formatting-mode
string
variable:timeout
*variable:timeout
variable#n:timeout
```

Use the I *formatting-mode* parameter with pipes. The I parameter lets you issue a timed **READ** for a named pipe without losing any data that can occur in a partial record that follows an <ENDOFFILE> error. When you use this parameter on a **READ**, the **READ** ignores <ENDOFFILE> messages.

The value of the I *formatting-mode* is "off" by default. If you include this parameter in a **READ** command without a timeout, your process hangs until there is data to process.

4.1.5 CLOSE Command for Interprocess Communication

If you create a child process using **OPEN** with a "Q" (/QUEUE) parameter code, the child process may survive a **CLOSE** operation on the device. Survivability of a queued interprocess communications pipe is platform-dependent. On UNIX systems the child process always survives the **CLOSE**. On Windows systems the survival of the process depends upon how old the process is. A child process that has just been initiated does not survive a **CLOSE** operation, but once a child process is fully established it survives a **CLOSE**.

On UNIX systems, you can specify the how long the **CLOSE** command should wait when closing a piped command device. The timeout default is 30 seconds. You can modify this default by specifying the **OPEN** command *closetimeout* positional argument. You can override the default or specified timeout for a **CLOSE** command by specifying the optional "I" positional argument. The "I" argument specifies immediate close (close after 1 second). The **CLOSE** syntax is as follows:

```
CLOSE cpipename:"I"
```

4.1.6 Using Named Pipes to Communicate with Visual Basic

On Windows, use named pipes in Caché as you would use TCP devices, but use the device name "|NPIPE|nnn" instead of "|TCP|nnn". The **OPEN** arguments are as follows:

```
OPEN "|NPIPE|3":(server:pipename)
```

where *server* is the Windows NT machine name, and *pipename* is the name of the pipe that it is to be connected to. Windows 95/98 machines cannot be named pipe servers, but can only connect to them.

To connect to a local pipename, use **"."** (a quoted period) as a server. To create a pipe (as a server), use **""** (quotes without content) as the server name. The following are all valid *server* names:

ObjectScript

```
OPEN "|NPIPE|3":(".":"localpipe")
OPEN "|NPIPE|3":("mother":"test")
OPEN "|NPIPE|3":("":"info")
```

A server can open a named pipe and immediately issue a write before the client side has opened the same named pipe. The write operation will hang until the client side opens the named pipe. A user can interrupt the hang by issuing a Control-C.

Once open, a pipe acts like an ordinary device. On the server side, clients can be disconnected as in TCP with:

ObjectScript

```
USE "|NPIPE|3":"DISCONNECT"
```

Alternatively:

ObjectScript

```
USE "|NPIPE|3" WRITE *-2
```

4.1.6.1 OPEN Command Keywords

The following table describes the keywords for controlling named pipes with only the **OPEN** command.

Table 4–3: OPEN Command Keywords for Named Pipes

Keyword	Default	Description
/HOSTNAME=*str* *or* /HOS=*str*	The default is `""` (quotes without content), which opens the pipe as a server.	Corresponds to the server positional parameter, which specifies the Windows NT workstation/server name. It is not necessary to specify this keyword when opening the pipe as a server. Use `"."` (a quoted period) to connect to a local pipename.
/IBUFSIZE=*n* *or* /IBU=*n*	2048	Specifies the size of the named pipe input buffer that holds data received from the pipe but not yet delivered to the application.
/INSTANCES=*n* *or* /INS=*n*	1	Specifies the maximum number of instances allowed for the named pipe. A value greater than 1 allows more than one server to open an instance of the named pipe, so that more than one client at a time can be served.
/OBUFSIZE=*n* *or* /OBU=n	2048	Specifies the size of the output buffer used by the operating system. This buffer size is advisory, since the operating system sizes the buffer according to system-imposed constraints.
/PIPENAME=*str* *or* /PIP=*str*	No default	Corresponds to the *pipename* positional parameter which specifies the name of the pipe.

4.2 Communication Between Caché Processes

Interjob communication (IJC) devices are a set of special device numbers that let you transfer information between two or more Caché processes. The processes can be either jobbed processes or interactive processes.

IJC devices work in pairs. You can have up to 256 IJC device pairs. You use even-numbered devices, called receivers, to read data. You use odd-numbered devices, called transmitters, to write data. Attempts to read from a transmitter or write to a receiver result in a <NODEV> error.

You issue I/O commands to IJC devices, just as to any other device. After issuing **OPEN** and **USE** commands to the device, a process can issue:

- **READ** commands to a receiver device
- **WRITE** commands to a transmitter device

Only one process at a time can have a device open.

Pairs are based on relative order as mapped in the Caché Device Table, which you can view and edit using the configuration options of the Management Portal.

Each pair of devices is associated with an IJC memory buffer. When a process issues a **WRITE** command to any odd-numbered IJC device, Caché writes the data into the buffer for that device pair. When another process issues a **READ** command to the even-numbered device from that pair, Caché reads the data from the same buffer.

Written data is buffered in memory in first-in, first-out fashion. If a **READ** occurs while the device is empty, the process that issued it suspends until another process issues a corresponding **WRITE**. A **WRITE** that occurs while the buffer is full suspends until another process reads from that buffer.

After you write a message to the buffer, it remains there until it is read, even if you close the transmitter. Several users can issue **OPEN**, **USE**, **WRITE**, and **CLOSE** commands to a transmitter, one at a time in turn. Subsequent **READ** commands get all of the messages in the order in which they were written.

4.2.1 Specifying Memory Buffers for Interjob Communication Devices

The system manager can configure the IJC buffers using the Management Portal. Select **[System] > [Configuration] > [Advanced Memory Settings]**. The two parameters that can be set are:

- **ijcnum**: The maximum number of IJC devices. The range is from 0 through 256. The default is 16. If you edit this setting, you must restart Caché to apply the change.
- **ijcbuff**: The maximum size (in bytes) of each IJC buffer. The range is from 512 through 8192. The default size is 512 bytes. If you edit this setting, you must restart Caché to apply the change.

Each IJC device corresponds to one IJC buffer of the size specified in **ijcbuff**. You can write a message of length **ijcbuff** minus 1.

4.2.1.1 Disabling Interjob Communication Buffers

If you will not be using IJC devices, you can set the maximum number of IJC devices (**ijcnum**) to 0 to avoid tying up memory.

4.2.2 Interjob Communication Device Numbers

Interjob communication devices are automatically defined numbered by Caché. Their actual identification numbers depends on the maximum number of IJC buffers configured on the system.

The table below gives the ranges of IJC device numbers that are available on your system, depending on the number of IJC buffers that you have allocated.

For example, if you allocate 8 IJC buffers, then device numbers from 224 through 239 are defined on the system (even numbers for **READ** devices and odd numbers for **WRITE** devices).

As another example, if you allocate 94 IJC buffers, then the following range of device numbers are defined: 224 through 255, 64 through 199, 4 through 19, and 2048 through 2051. You can use any even/odd number pairs with **OPEN**, **USE**, **READ**, **WRITE**, and **CLOSE** commands.

Table 4–4: IJC Device Numbers

Buffers Allocated	READ Device #	WRITE Device #
1	224	225
2	226	227
3	228	229
...	...	
15	252	253
16	254	255
17	64	65

Buffers Allocated	READ Device #	WRITE Device #
18	66	67
...	...	...
83	196	197
84	198	199
85	4	5
86	6	7
87	8	9
88	10	11
89	12	13
90	14	15
91	16	17
92	18	19
93	2048	2049
94	2050	2051
95	2052	2053
...	...	...
254	2370	2371
255	2372	2373
256	2374	2375

4.2.3 I/O Commands for IJC Devices

You use all of the standard I/O commands with IJC devices: **OPEN**, **USE**, **READ**, **WRITE**, and **CLOSE**.

4.2.3.1 OPEN Command

The **OPEN** command reserves interjob communication devices for your use.

Syntax

```
OPEN device::timeout
```

where:

device	A device number from the table above. **OPEN** an even-numbered device to issue **READ** commands. **OPEN** an odd-numbered device to issue **WRITE** commands. For two processes to communicate, they must open a set of device pairs.
timeout	*Optional* — A positive integer whose value in seconds is the longest time Caché waits for an **OPEN** to finish. If you specify 0, the **OPEN** returns control to the process immediately.

This example shows how two processes communicate by opening separate devices for reading and writing:

```
Process A                          Process B
OPEN 227 USE 227 WRITE "MSG_1"
WRITE "MSG_2"                       OPEN 226 USE 226 READ X
CLOSE 227                           CLOSE 226
OPEN 224 USE 224 READ X             WRITE X
CLOSE 224                           MSG_1
WRITE X                             .
MSG_3                               .
                                    .
                                    OPEN 225 USE 225 WRITE "MSG_3"
                                    CLOSE 225
```

Process A begins by opening device 227 and writing MSG_1 to it. Caché writes this message into the buffer shared by devices 226 and 227. Process A then writes a second message to the same buffer. Now Process B opens companion device 226 and reads the first message (MSG_1) from the buffer.

Now Process A wants to read a message, so it must open a different device, 224. Because the buffer for this device and its companion, 225, is currently empty, Process A waits until Process B opens device 225 and writes MSG_3 to it. After Caché places this message in the buffer shared by devices 224 and 225, the **READ** command to device 224 succeeds.

5

TCP Client/Server Communication

This chapter describes how to set up remote communication between Caché processes using TCP/IP. For local communication between processes using pipes or using Interjob Communication (IJC) devices, refer to the Local Interprocess Communication chapter of this manual.

Caché supports two Internet Protocols (IP): TCP and UDP. These Internet Protocol allow Caché processes to communicate with processes on local or remote systems, whether or not those processes are running Caché.

- TCP: the Caché Transmission Control Protocol (TCP) binding. Establishes a two-way connection between a server and a single client. Provides reliable byte stream transmission of data with error checking and correction, and message acknowledgement.
- UDP: the Caché User Datagram Protocol (UDP) binding. Provides two-way message transfer between a server and a large number of clients. UDP is not connection-based; each transmission of data packets is an independent event. Provides fast and lightweight data transmission for local packet broadcasts and remote multicasting. Inherently less reliable than TCP. Does not provide message acknowledgement. For details, refer to the UDP Client/Server Communication chapter of this manual.

The goal of TCP binding is to hook Caché up to a widespread networking standard so that basic features of the underlying network protocol are available to Caché users through I/O commands.

The TCP/IP protocol allows systems to communicate even if they use different types of network hardware. For example, TCP, through an Internet connection, transmits messages between a system using Ethernet and another system using Token Ring. TCP controls the accuracy of data transmission. IP, or Internet Protocol, performs the actual data transfer between different systems on the network or Internet.

Using TCP binding, you can create both client and server portions of client/server systems. In the client/server type of distributed database system, users on one or more client systems can process information stored in a database on another system, called the server.

5.1 TCP Connections Overview

To create a client/server relationship between systems, you must follow a particular set of conventions:

- Your systems must be connected with appropriate networking hardware and software, including TCP/IP protocol software.
- Systems communicate with each other through a TCP port. The processes at both ends of the connection must use the same port number.

- You specify either the TCP port number, or the *devicename* of the device that represents it, as the device in Caché **OPEN**, **USE**, and **CLOSE** commands.

Using these conventions, the general procedure of establishing a TCP binding connection is:

1. The server process issues an **OPEN** command to a TCP device.
2. The server process issues a **USE** command, followed by a **READ** command, awaiting input from the client process. The server must be listening before a client can establish a connection. The initial **READ** command completes when the client has opened the connection and sent some data. You can include the "A" mode parameter in the **OPEN** command to make the initial **READ** complete as soon as the server accepts the connection.
3. The client process issues an **OPEN** command that specifies the TCP device to which it is connecting.
4. The client process issues a **USE** command followed by a **WRITE** command to complete the connection. Caché copies all characters in the **WRITE** command(s) to a buffer. It does not write them to the network until you issue a `WRITE !` or `WRITE #` command to flush the buffer.
5. After the server has read the characters that the client sent in its first **WRITE** command, both sides can continue to issue **READ** and **WRITE** commands. There is no further restriction on the order of these commands to the same port.
6. Either side can initiate the closing of a connection with the **CLOSE** or **HALT** command. Closing the client side first is preferable. If the server needs to disconnect so that it can accept a connection from another client process, it can instead issue either a `WRITE *-2` command, or for compatibility with older versions, a **USE** command with the "DISCONNECT" option, and follow either one with a **READ** command.

Note: This procedure assumes that both the client and server are Caché processes (though either process can be a non-Caché process).

The following sections detail how to use Caché I/O commands to create a TCP binding between client and server processes.

5.2 OPEN Command for TCP Devices

Both server and client processes use the ObjectScript **OPEN** command to initiate a connection. The server completes the connection by issuing a READ command, which receives the client **OPEN** command and first data transmission.

Note: If you issue an **OPEN** command on a TCP device that has already been opened, this second **OPEN** command is treated as a USE command. That is, the *hostname* and *port* parameters are ignored (retaining the first **OPEN** command values) and the *mode* and *terminators* parameters are updated.

5.2.1 Using the OPEN Command

The **OPEN** command reserves a TCP binding device for your use. The syntax is:

```
OPEN devicename:parameters:timeout:mnespace
```

where

devicename	A string of the form |TCP| followed by some number of numeric digits. The numeric portion of the device name is called the device identifier. If the port number is not specified in the **OPEN** parameters, this device identifier must be a unique five-digit TCP port number. If the port number is specified in the **OPEN** parameters (which is the preferred practice), this device identifier can be any unique number (up to a maximum of 2147483647), so long as all the TCP device names used by a single job are distinct.
parameters	*Optional* — A series of one or more device parameters, enclosed by parentheses and separated by colons (:). If a parameter is omitted, specify the colon separator for the missing parameter. (For a server-side **OPEN** the first parameter is always omitted.) The specific parameters are described below. If you specify only the first parameter (*hostname*), you can omit the parentheses. For example, the client-side open: `OPEN "|TCP|7000":"127.0.0.1":10`. If you specify no parameters, you can omit the parentheses, but you must retain the colon as a separator character. For example, the server-side open: `OPEN "|TCP|7000"::10`.
timeout	*Optional* — Maximum number of seconds Caché attempts to open the TCP device. If it does not succeed within this interval, it sets **$TEST** to 0 and returns control to the process. If it succeeds, it sets **$TEST** to 1. Including a timeout in **OPEN** commands from the client prevents the client system from hanging if it tries to open a connection while the server is busy with another client. The server can have only one connection open at a time.
mnespace	*Optional* — Supported as it is for all ObjectScript **OPEN** commands. There is no predefined mnemonic space for TCP bindings.

If you omit an **OPEN** argument, you can indicate its absence by specifying the colon separator.

The *timeout* argument, though optional, is strongly recommended because the success or failure of **OPEN** is indicated by the value of the $TEST special variable, and **$TEST** is only set if *timeout* is specified. **$TEST** is set to 1 if the open attempt succeeds before the timeout expires; if the timeout expires, **$TEST** is set to 0.

If a TCP connection attempt fails on WIndows systems, the TCP connection error is written to the Caché system error log (see Caché System Error Log in the "Monitoring Caché Using the Management Portal" chapter of the *Caché Monitoring Guide*), for example, error code 10061 = WSAECONNREFUSED.

The following is an example of a client-side **OPEN**, where 7000 is the port number and "127.0.0.1" is the *parameters* argument (the *hostname*, specified as an IPv4 address):

ObjectScript

```
SET dev="|TCP|7000"
OPEN dev:("127.0.0.1":7000)
```

5.2.1.1 hostname Parameter

The *hostname* parameter is required for a client-side **OPEN**. The client-side *parameters* argument may be just the *hostname*, or the *hostname* followed by other colon-separated parameters. If you specify just the *hostname* parameter, you can omit the *parameters* parentheses.

The server-side *parameters* argument omits the *hostname*.

The *hostname* can be either the name of an IP host (from the local system's database of remote hosts) or an IP address in either IPv4 or IPv6 protocol format. Because these protocols are incompatible, both the server and the client must use the same Internet protocol or the transmission will fail.

An IPv4 address has the following format. *n* is a decimal integer in the range 0 through 255:

```
n.n.n.n
```

An IPv6 address has the following full format. *h* is a hexadecimal number with four hexadecimal digits:

```
h:h:h:h:h:h:h:h
```

Commonly, IPv6 addresses are abbreviated by eliminating leading zeros and replacing consecutive sections of zeros with a double colon (::); only one double colon may be used in an IPv6 address. By using IPv4 abbreviation rules, you can specify the IPv6 loopback address as `"::1"` (meaning that the first seven consecutive *h* sections all have the value 0000, and the leading zeros from the eighth section are eliminated).

Further details on IPv4 and IPv6 formats can be found in the section "Use of IPv6 Addressing" in the chapter "Server Configuration Options" in the *Caché Programming Orientation Guide*.

5.2.1.2 Supported Parameters

The *parameters* argument can be in either of the following formats:

```
hostname
```

```
(hostname{:port{:mode{:terminators{:ibufsiz{:obufsiz{:queuesize{:keepalivetime}}}}}}})
```

The parameters within the *parameters* argument are as follows:

Parameter	Meaning
hostname	*Optional* — Either the name of an IP host, an IP address in IPv4 protocol format, or an IP address in IPv6 protocol format. Specified as a quoted string. A *hostname* is required for a client-side **OPEN**; omitted (represented by a placeholder colon) for a server-side **OPEN**.
port	*Optional* — If present, this is the TCP port number to use for the connection. If this port number is null or omitted, then the port number is derived from the numeric portion of the *devicename*. This parameter can either be a decimal port number or a service name, which is submitted to the local system's TCP service name resolver.

Parameter	Meaning
mode	*Optional* — A string of letter code characters enclosed in quotes. Letter codes may be specified in any order; because Caché executes them in left-to-right order, interactions between letter codes may dictate a preferred order in some cases. The default is packet mode. A *mode* string can consist of one or more of the following letter codes: • A—Accept mode. If A is on, the initial read on the server terminates with a zero-length string as soon as the connection from the client job is accepted. If A is off, the read blocks until the timeout is reached, or until data is available, whichever occurs first. • C—See Carriage Return Mode below. • D—See Monitoring for Disconnect Mode below. • E—See Escape Sequence Processing Mode below. • G—Causes the *port* parameter to be interpreted as the socket descriptor of an already opened data socket. • M—Standard Caché device in stream mode. This mode is a shorthand for invoking the "PSTE" set of options. It yields a device that acts like a standard Caché device that can be used to pass arbitrary lines of data in both directions. You turn on stream mode so that you can send or receive any arbitrary sequence of strings, without overrunning the buffers. Line feeds are added to output and stripped from input. **READ** commands block until one of the following occurs: a terminator character is seen, the timeout is reached, or the read length specified has been filled. • P—Pad output with record terminator characters. When this mode is set, `WRITE !` sends LF (line feed) and `WRITE #` sends FF (form feed), in addition to flushing the write buffer. The `WRITE *-3` command can be used to initiate the sending of buffered data without inserting any characters into the data stream. Note that `WRITE *-3` just flushes the write buffer without sending any terminator character, and thus does not signal the recipient program that the data is complete. `WRITE *-3` is more commonly used in Wait (W) mode, which does not require a terminator. • Q—See Send Immediate Mode below. • S—See Stream Mode below. • T—Standard terminators on input. When this is set, the CR, LF, and FF control characters function as read terminators. • W—Wait mode. In this mode, `WRITE !` and `WRITE #` commands will not cause a TCP device to flush the network output buffers. Wait mode causes a TCP device to wait until the next `WRITE *-3` command to flush the buffers and transmit the data.
terminators	*Optional* — A list of up to eight user terminator characters that will terminate reads on the TCP binding device. If you specify both T mode and *terminators* at the same time, T mode is ignored.
ibufsiz	*Optional* — Input buffer size. Internally, characters that have been read from the network but not yet delivered to the Caché program are buffered in a data area that can hold *ibufsiz* bytes.

Parameter	Meaning
obufsiz	*Optional* — Output buffer size. The maximum amount of data the TCP device can buffer between successive "SEND" operations. A SEND operation means to send the buffered data out to the network. `WRITE !`, `WRITE #`, and `WRITE *-3` commands can generate SEND operations. When S mode is specified, SEND operations are generated automatically to send the contents of the output buffer whenever it gets too full. When done creating a message, however, the programmer must still use one of the SEND operations to make sure the message is sent. When S mode is not specified, if a **WRITE** operation would place enough data in the buffer to exceed the output buffer size, then a <WRITE> error occurs. Note that attempting to write a string that is in itself longer than the output buffer size always fails.
queuesize	*Optional* — An integer that specifies how many client jobs can queue for a connection to the server. Used for server-side **OPEN** only. The default is 5. The maximum value depends on the TCP implementation, but cannot exceed 1000.
keepalivetime	*Optional* — (Windows, AIX, and Linux only) Allows you to set a keepalive timer for this device that is different than the system default. Specify an integer number of seconds to keep alive the TCP connection. Valid values range from 30 to 432000. (432000 seconds is 5 days.) A value less than 30 defaults to 30. If omitted or set to 0, the system-wide default keepalive timer is used. See the /KEEPALIVE keyword option for further details. The keepalive timer does not necessarily start timing when the TCP device is opened. It typically begins timing when the connection has been established. That is, when the initial read-for-connect has completed successfully.

5.2.1.3 Packet Mode

Packet mode is the default if no *mode* is specified. If stream mode is disabled, the mode defaults to packet mode.

In packet mode **READ** commands complete as soon as there is some data to return. Packet mode allows you to build an entire TCP segment in the output buffer, and then send it all at one time by issuing a `WRITE *-3` or `WRITE !` command.

If you issue `WRITE *-1` to initiate a TCP SEND operation when there are no characters to be sent, you receive a <WRITE> error. If you issue **WRITE** of an empty string, you receive a <COMMAND> error.

The maximum size of the string you can send in packet mode is 1024 characters. If you exceed this limit without flushing the buffer, you receive a <WRITE> error.

Because TCP/IP ignores records with a length of 0, you receive a <WRITE> error if you flush the write buffer when there are no characters in it.

A **WRITE** command from server to client before the server has received a connection request produces a <WRITE> error on the server.

5.2.1.4 Carriage Return Mode (C mode)

This mode modifies processing of carriage returns on input and output.

On Output, `WRITE !` generates "CR LF" and `WRITE #` generates "CR FF".

On input, with T mode enabled, the server tries to record an adjacent CR and LF or an adjacent CR and FF as a single terminator in **$ZB**. CR and LF are processed as separate terminators if they do not arrive within a short interval of each other. By default, the interval is 1 second.

5.2.1.5 Monitoring for Disconnect Mode (D mode)

This mode turns on or off asynchronous disconnect monitoring. This mode is activated by specifying the "D" mode character, or the /POLL or /POLLDISCON keyword parameter. When you specify +D, TCP disconnect monitoring is activated; when you specify –D, TCP disconnect monitoring is deactivated.

While activated, Caché polls the TCP connection roughly every 60 seconds. When it detects a disconnect, Caché issues a <DISCONNECT> error. Disconnect detection does not occur in idle jobs, such as a job suspended by a **HANG** command or a job waiting on a **READ** operation. Caché suspends all disconnect monitoring during a rollback operation to prevent a <DISCONNECT> error being issued. Caché resumes disconnect monitoring once the rollback concludes. This suspension applies both to a current TCP device with disconnect monitoring activated, and to a current device without disconnect monitoring that is connected to a TCP device with disconnect monitoring activated.

You can also check for TCP disconnect by using the **Connected()** method of the %SYSTEM.INetInfo class.

5.2.1.6 Escape Sequencing Processing Mode (E mode)

When the E mode is set, escape sequences in the input stream are parsed and placed into the **$ZB** special variable. Escape sequences must be 15 characters or less and must match the following syntax:

```
esc_seq::=type1 | type2
```

where:

```
type1 ::= '['['0':'?']*['':'/']*{'@':DEL}
type2 ::= [';'|'?'|'O']['':'/']*{'0':DEL}
```

The syntactic symbols used here mean:

:	x:y means a specified range of characters from *x* through *y* in the ASCII sequence.
|	x|y means specify either *x* or *y*.
[ ]	Specify zero or one members of the specified set.
[ ]*	Specify zero, one, or more members of the specified set.
{ }	Specify exactly one member of the specified set.

When Caché sees an ESCAPE, it waits up to 1 second for the rest of the escape sequence to arrive. If the escape sequence does not match this syntax, or if it is longer than 15 characters, or if a valid escape sequence does not arrive within 1 second, Caché places the partial escape sequence in **$ZB** and sets the "BADESC" bit (256) in **$ZA**.

5.2.1.7 Send Immediate Mode (Q mode)

In send immediate mode, each **WRITE** command is output as its own packet. If you are not using send immediate mode, you must either include a terminator or issue the command `WRITE *-3` to output a packet.

This mode is entered by specifying the "Q" mode character, or the /SENDIMMEDIATE (or /SEN) keyword parameter. To turn this option off, specify either of the following:

ObjectScript

```
USE TCPDEVICE:(/SEN=0)
USE TCPDEVICE:(::"-Q")
```

To turn this option back on, specify either of the following:

ObjectScript

```
USE TCPDEVICE:(/SEN=1)
USE TCPDEVICE:(::"+Q")
```

Send Immediate Mode, which creates one packet per write, is used in combination with /NODELAY mode, which immediately sends each packet as it is created. When both are on, the speed of transmission of a single burst of data is maximized. This is useful when timely delivery of each unit of data is critical, for example, in transmitting mouse movements. When both are off, a packet may contain multiple writes, and a transmission may contain multiple packets. This reduces network traffic and improves overall performance. The default for Send Immediate Mode is off. The default for /NODELAY mode is on.

5.2.1.8 Stream Mode (S mode)

In stream mode, Caché does not attempt to preserve TCP message boundaries in the data stream. On sending, if the data does not fit in the message buffer, Caché flushes the buffer before placing the data in it.

On receiving, data up to the maximum string length can be received. All reads wait for the full timeout for terminators to be reached or for the buffer to become full. When this mode is disabled (the default), you are in packet mode.

Jobbed processes that inherit TCP devices are automatically set to Stream format. You can reset the format with the **USE** command.

5.2.1.9 Buffer Sizes

The *ibufsiz* and *obufsiz* parameters for TCP devices specify the sizes of the internal Caché buffers for TCP input and output. They can take values between 1KB and 1MB on all supported platforms. However, operating system platforms may use different sizes for their own input and output buffers. If the operating system platform buffer is smaller than the Caché buffer (for example, 64KB vs 1MB), performance may be affected: a WRITE operation may require several trips to the OS to send the entire Caché buffer; a READ operation may return smaller chunks that are limited by the OS buffer size. For optimal performance, a user should experiment with the current OS to determine which values for *ibufsiz* and *obufsiz* produce optimal results.

5.2.2 Server-Side OPEN Command

When the server-side **OPEN** is processed, it establishes a TCP socket and listens on the socket for incoming connection requests on the appropriate port number. The port number is either specified explicitly in the parameter list, or derived from the numeric portion of the *devicename*. The **OPEN** returns immediately after the socket has been set up to listen.

If the **OPEN** does not succeed, another process may already be listening for connection requests on that port number.

The following example of a server-side **OPEN** shows a Caché-like device specification that allows reading and writing of terminated strings up to the maximum string size, and uses maximum length read and write operations to consolidate use of the TCP channel.

ObjectScript

```
OPEN "|TCP|4":(:4200:"PSTE"::32767:32767)
```

The *parameters* argument in this example is as follows: because this is a server-side **OPEN**, the first parameter (*hostname*) is omitted. The second parameter explicitly specifies the port number (4200). The third parameter is the *mode* code characters. The fourth parameter (*terminators*) is omitted. The fifth parameter is the input buffer size. The sixth parameter is the output buffer size.

The following example is backward compatible with versions prior to Open M [ISM] Version 5.6. The port number is not specified as a parameter; it is derived from the numeric portion of the *devicename*. This example opens port 4200 with no specified parameters and a timeout of 10 seconds:

ObjectScript

```
OPEN "|TCP|4200"::10
```

A server-side **OPEN** has default input buffer size (*ibufsiz*) and output buffer size (*obufsiz*) parameter values of 1,048,576 bytes (1 MB).

A server-side **OPEN** supports the optional *queuesize* parameter, and the optional "G" mode parameter. These options are not available to a client-side **OPEN**.

A server-side **OPEN** supports the optional /CLOSELISTEN keyword parameter. This option is not available to a client-side **OPEN**.

5.2.3 Client-Side OPEN Command

A client-side **OPEN** command differs from the server-side **OPEN** command in only one respect: the first device parameter must specify the host to which you are connecting. To specify the host, you include either a name that the client recognizes as a host, or an Internet address.

The **OPEN** succeeds as soon as the connection is established. At this point, you can read or write to the TCP device. However, if the server side of the connection is another Caché process, the server does not complete its side of the connection until some data has been sent from the client to the server with the **WRITE** command. Therefore, you must issue a **WRITE** command before you issue any **READ** commands.

For details, see the section "WRITE Command for TCP Devices".

Some examples of client-side **OPEN** commands are:

ObjectScript

```
OPEN "|TCP|4":("hal":4200::$CHAR(3,4)):10
```

This command opens a connection to host hal on port 4200. It specifies no mode string. It specifies two terminators (ASCII $CHAR(3) and $CHAR(4)), and default input and output buffer sizes. It specifies a timeout of 10 seconds.

The following command is the same as the previous one, except that the destination is an explicit IP address in IPv4 format.

ObjectScript

```
OPEN "|TCP|4":("129.200.3.4":4200::$CHAR(3,4)):10
```

Further details on IPv4 and IPv6 formats can be found in the section "Use of IPv6 Addressing" in the chapter "Server Configuration Options" in the *Caché Programming Orientation Guide*.

The following command connects to time-of-day server on remote host "larry" and prints the remote host's time of day in ASCII format on the principal input device. It uses the service name `daytime`, which the local system resolves to a port number:

ObjectScript

```
OPEN "|TCP|4":("larry":"daytime":"M")
USE "|TCP|4"
READ x
USE 0
WRITE x
```

The following command sets *x* to "hello":

ObjectScript

```
OPEN "|TCP|4":("larry":"echo":"M")
USE "|TCP|4"
WRITE "hello",!
READ x
```

The following command is backwards compatible with versions prior to Open M [ISM] Version 5.6. It opens a connection to Internet address 128.41.0.73, port number 22101, with a 30-second timeout.

ObjectScript

```
OPEN "|TCP|22101":"128.41.0.73":30
```

5.2.4 OPEN and USE Command Keywords for TCP Devices

You can either use positional parameters (as described above) or keyword parameters. The following table describes the keywords for controlling TCP devices with both **OPEN** and **USE** commands. There are additional OPEN-only keywords (described later in this chapter) that can only be specified in the **OPEN** command. All keyword parameters are optional.

Table 5–1: OPEN and USE Command Keywords for TCP Devices

Keyword	Default	Description
/ABSTIMEOUT[=1]	0	Specifies read timeout behavior. Determines whether TCP should reinitialize the timeout period when data is received. If /ABSTIMEOUT=0 (the default) timeout is reset to its original value each time data is received. If /ABSTIMEOUT or /ABSTIMEOUT=1 the timeout period continues to count down while data is received.
/ACCEPT[=*n*] *or* /ACC[=*n*]	0	Corresponds to the "A" mode parameter character, which specifies that the initial read on the server terminates with a zero length string as soon as the connection from the client job is accepted. /ACCEPT and /ACCEPT=*n* for nonzero values of *n* enable A mode. /ACCEPT=*n* for a zero value of *n* disables A mode.
/CLOSEFLUSH[=*n*]	1	Specifies handling of data remaining in the output buffer when the device is closed. /CLOSEFLUSH and /CLOSEFLUSH=*n* for nonzero values of *n* flushes remaining data. /CLOSEFLUSH=*n* for a zero value of *n* discards remaining data.
/CRLF[=*n*]	0	Corresponds to the "C" mode parameter character, which modifies processing of carriage returns on input and output. /CRLF and /CRLF=*n* for nonzero values of *n* enable C mode. /CRLF=*n* for a zero value of *n* disables C mode.
/ESCAPE[=*n*] *or* /ESC[=*n*]	0	Corresponds to the "E" mode parameter character, which specifies that escape sequences in the input stream are parsed and placed into $ZB. /ESCAPE and /ESCAPE=*n* for nonzero values of *n* enable E mode. /ESCAPE=*n* for a zero value of *n* disables E mode.

Keyword	Default	Description
/GZIP[=*n*]	1	Specifies GZIP-compatible stream data compression. /GZIP or /GZIP=*n* (for nonzero values of *n*) enables compression on WRITE and decompression on READ. /GZIP=0 disables compression and decompression. Before issuing /GZIP=0 to disable compression and decompression, check the $ZEOS special variable to make sure that a stream data read is not in progress. /GZIP compression has no effect on I/O translation, such as translation established using /IOTABLE. This is because compression is applied after all other translation (except encryption) and decompression is applied before all other translation (except encryption). For further information on WRITE with compressed data, refer to WRITE Control Characters in this chapter.
/IOTABLE[=*name*] *or* /IOT[=*name*]	If *name* is not specified, the default I/O translation table for the device is used.	Establishes an I/O translation table for the device.
/KEEPALIVE=*n*	system default	(Windows, AIX, and Linux only) Allows you to set a keepalive timer for this device that is different than the system default. An integer that specifies the number of seconds to keep alive the TCP connection. Same as the *keepalivetime* positional parameter. Valid values range from 30 to 432000. (432000 seconds is 5 days.) A value less than 30 defaults to 30. If omitted or set to 0, the system default is used. This setting can be disabled using /NOKEEPALIVE; once disabled, it cannot be re-enabled until this TCP device is closed.
/NODELAY=*n*	1	Specifies whether packets should be bundled or sent individually. If /NODELAY=1 (the default) each packet is immediately transmitted. If /NODELAY=0 the TCP driver bundles packages together using an optimization algorithm. This can cause a slight transmission delay for an individual packet, but by reducing network traffic it can improve overall performance. /NODELAY has no corresponding mode parameter character. Use of /NODELAY should be coordinated with use of /SENDIMMEDIATE.

Keyword	Default	Description
/NOKEEPALIVE		If specified, the system-wide TCP keepalive timer is disabled for this device. Cache enables this timer by default when opening any TCP device; issuing the /NOKEEPALIVE option on OPEN or USE overrides this default. If /KEEPALIVE has been used to set a non-default keepalive timer, /NOKEEPALIVE disables that keepalive timer. Once you disable a keepalive timer there is no way to re-enable it until the TCP device is closed. See /KEEPALIVE.
/NOXY[=*n*]	0	No $X and $Y processing: /NOXY or /NOXY=*n* (for nonzero values of *n*) disables $X and $Y processing. This option can improve performance when device $X/$Y is not used, for example with CSP. It can substantially improve performance of READ and WRITE operations. This option is the default setting for superserver slave jobs. When /NOXY=1, the values of the $X and $Y variables are indeterminate, and margin processing (which depends on $X) is disabled. /NOXY=0 enables $X and $Y processing; this is the default. /TCPNOXY is a deprecated synonym for /NOXY.
/PAD[=*n*]	0	Corresponds to the "P" mode parameter character, which specifies that output is padded with record terminator characters when `WRITE !` (LF terminator) or `WRITE #` (FF terminator) is executed. /PAD and /PAD=*n* for nonzero values of *n* enable P mode. /PAD=*n* for a zero value of *n* disables P mode.
/PARAMS=*str* *or* /PAR=*str*	No default	Corresponds to the *mode* positional parameter. (It provides a way to specify a mode string in a position-independent way.)
/POLL[=*n*] *or* /POLLDISCON[=*n*]		Corresponds to the "D" mode parameter character, which specifies asynchronous monitoring for disconnect. /POLL or /POLL=1 corresponds to +D. /POLL=0 corresponds to -D.
/PSTE[=*n*]	0	Corresponds to the "M" mode parameter character, which is a shorthand way to specify the P, S, T and E mode parameter characters. /PSTE and /PSTE=*n* for nonzero values of *n* enable P, S, T and E modes. /PSTE=*n* for a zero value of *n* disables these modes.
/SENDIMMEDIATE[=*n*] *or* /SEN[=*n*]	0	Corresponds to the "Q" mode parameter character, which specifies Send Immediate Mode.

Keyword	Default	Description
/SSL="*cfg*[|*pw*] [|*DNShost*]" *or* /TLS="*cfg*[|*pw*] [|*DNShost*]"	No default	From a client, specifies that the device attempts to negotiate an SSL/TLS-secured connection according to the client's specified configuration and server requirements. When securing a socket as a server, specifies that the server requires a SSL/TLS-secured connection according to the server's specified configuration and any client requirements. *cfg* specifies the name of the configuration for the connection or socket. *pw* specifies the optional private key file password. *DNShost* specifies the fully qualified DNS hostname of a specific server, for use with the Server Name Indication (SNI) TLS extension. See below for details. This configuration name is used only the first time I/O is performed after the **OPEN** or **USE** command. Subsequent invocations are ignored. `/SSL=""` or `/TLS=""` is ignored. For more information, see the "Using SSL/TLS with Caché" chapter in the *Caché Security Administration Guide*. IMPORTANT: The ability to include a password when opening a new or securing an existing TCP connection using SSL/TLS is for real-time interactive use only. You should *never* store a private key password persistently without protecting it. If you need to store such a password, use the PrivateKeyPassword property of the Security.SSLConfigs class.
/STREAM[=*n*] *or* /STR[=*n*]	0	Corresponds to the "S" mode parameter character, which specifies a stream mode of handling data that does not preserve TCP message boundaries. /STREAM and /STREAM=*n* for nonzero values of *n* enable S mode. /STREAM=*n* for a zero value of *n* disables S mode.
/TCPNOXY		Deprecated. A synonym for /NOXY.
/TCPRCVBUF=*n*	Default receive buffer size	Set receive queue buffer size, in bytes. Can be used to increase the buffer size from the default value to support TCP protocol large windows. Large windows improve performance over links with long latencies or very high bandwidth. For appropriate values, consult your OS/hardware documentation.
/TCPSNDBUF=*n*	Default send buffer size	Set send queue buffer size, in bytes. Can be used to increase the buffer size from the default value to support TCP protocol large windows. Large windows improve performance over links with long latencies or very high bandwidth. For appropriate values, consult your OS/hardware documentation.

Keyword	Default	Description
/TERMINATOR=*str* *or* /TER=*str*	No default	Corresponds to the *terminators* positional parameter, which establishes user-defined terminators.
/TMODE[=*n*] *or* /TMO[=*n*]	0	Corresponds to the "T" mode parameter character, which specifies CR, LF, and FF as standard read terminators. /TMODE and /TMODE=n for nonzero values of *n* enable T mode. /TMODE=*n* for a zero value of *n* disables T mode.
/TRANSLATE[=*n*] *or* /TRA[=*n*]	1	/TRANSLATE or /TRANSLATE=*n* for nonzero values of *n* enable I/O translation for the device. /TRANSLATE=*n* for a zero value of *n* disables I/O translation for the device.
/WAIT[=*n*]	0	Corresponds to the "W" mode parameter character, which causes output buffers not to be flushed by the `WRITE !` and `WRITE #` commands. Rather, flushing waits until the next `WRITE *-3` command. /WAIT and /WAIT=*n* for nonzero values of *n* enable W mode. /WAIT=n for a zero value of *n* disables W mode.
/WRITETIMEOUT[=*n*]	-1	Establishes a timeout (in seconds) for TCP write operations. If a write does not complete within *n* seconds, Caché issues a <TCPWRITE> error. If a <TCPWRITE> error is issued, your application should immediately close the TCP device to prevent data loss. Caché will not attempt a TCP write operation following a <TCPWRITE> error. The minimum *n* value is system-dependent. If *n* is smaller than the minimum timeout value for the platform, Caché uses the platform minimum. No *n* value should be less than 2. The default (-1) indicates no timeout is enforced.
/XYTABLE[=*name*] *or* /XYT[=*name*]	If *name* is not specified, the default $X/$Y action table for the device is used.	Establishes a $X/$Y action table for the device. See /NOXY.

5.2.4.1 SSL / TLS Components

The value of the TCP device OPEN or USE /SSL or /TLS keyword parameter as a quoted string. This string can have can have one, two, or three components, separated by the '|' character:

cfg	The name of the SSL Configuration to use for this connection. This component is required.
pw	*Optional* — The password for the local private key file. This is intended for interactive applications only, when a user is being prompted to enter the password at run time. It should not be used with a persistently stored password. Use the Security.SSLConfigs.PrivateKeyPassword property for persistent storage.
DNShost	*Optional* — For SSL clients only. Specify either the server-selected certificate (for Hostname Verification) or the fully qualified DNS hostname of a specific server (for Server Name Indication). If you omit *pw* you must specify the placeholder '|' character. Hostname Verification is a feature that allows the client to check that the certificate it receives from a server includes a field with the hostname that the client tried to connect to. This is for use by client applications, such as %Net.HttpRequest(), that want to verify that the server X.509 certificate contains a fully qualified server DNS hostname matching the server name in the URL, either in the subjectAltName extension or the Subject CN field. This allows clients to detect cases where a man in the middle attack uses a valid certificate for the wrong domain. Server Name Indication (SNI) is a feature that allows the client to submit the hostname it's asking for to the server. This allows a server which handles multiple domains to select one of its multiple certificates to return. The server can select one which will match hostname checking on the client.

The following are examples of valid /TLS keyword parameters:

```
/TLS="Client"
/TLS="Client|password"
/TLS="Client||www.intersystems.com"
/TLS="Client|password|www.intersystems.com"
```

5.2.5 OPEN-Only Command Keywords for TCP Devices

The following table describes the keywords for controlling TCP devices that can only be specified in the **OPEN** command. There are additional OPEN/USE keywords (described earlier in this chapter) that can be specified with either the **OPEN** or **USE** command. All keyword parameters are optional.

Table 5–2: OPEN-only Command Keywords for TCP Devices

Keyword	Default	Description
/BINDTO[=*address*]		Binds to a specified local address that is used when initiating connection. For client, this is the source address used when opening a TCP/IP connection from Caché. For server, this is the IP address that the Caché process will accept connections on when opening a TCP/IP connection. /BINDTO=*address* is used to control which network interface the connection will use. /BINDTO or /BINDTO="" deletes a previously specified address.
/CLOSELISTEN		(Server only) Prevents more than one remote connections to the listening port. If specified, the listen socket is closed after the first connection is accepted. Additional clients attempting to connect will time out on the **OPEN** command.

Keyword	Default	Description
/CONNECTIONS=*n* *or* /CON=*n*	5	Corresponds to the *queuesize* positional parameter, which determines how many client jobs can queue for a connection to the server.
/HOSTNAME=*str* *or* /HOS=*str*	No default	Corresponds to the *hostname* positional parameter, which is either the name of an IP host or an IP address in IPv4 or IPv6 address format. Further details on IPv4 and IPv6 formats can be found in the section "Use of IPv6 Addressing" in the chapter "Server Configuration Options" in the *Caché Programming Orientation Guide*.
/IBUFSIZE=*n* *or* /IBU[=*n*]	1024	Corresponds to the *ibufsiz* positional parameter, which specifies the size of the TCP input buffer that holds data read from the network, but not yet delivered to the application.
/OBUFSIZE=*n* *or* /OBU[=*n*]	1024	Corresponds to the *obufsiz* positional parameter, which specifies the size of the TCP output buffer that contains data that is held between successive "SEND" operations.
/PORT=*n*	No default	Corresponds to the *port* positional parameter, which is either the TCP port number or a service name to use for the connection.
/SOCKET=*n* *or* /SOC=*n*	No default	Corresponds to the "G" mode parameter character, which causes the *port* positional parameter to be interpreted as the socket descriptor of an already opened data socket. This keyword takes as its value that socket descriptor and is used instead of the /PORT=*n* keyword. (A socket descriptor is passed to ObjectScript from another programming environment (such as C) using the Caché Call-in or Call-out (**$ZF**) mechanisms.)

The following example shows a TCP/IP device being opened using keyword syntax:

ObjectScript

```
SET dev="|TCP|"_123
SET portnum=57345
OPEN dev:(/PSTE:/HOSTNAME="128.41.0.73":/PORT=portnum)
```

5.3 Current TCP Device

You can return the IP address and port number of the current TCP device using the methods of the %SYSTEM.TCPDevice class. You can list these methods using the **Help()** method, as follows:

ObjectScript

```
DO $SYSTEM.TCPDevice.Help()
```

You can display information about a specific method by specifying the method name in **Help()**, as shown in the following example:

ObjectScript

```
DO $SYSTEM.TCPDevice.Help("LocalAddr")
```

5.4 USE Command for TCP Devices

The **USE** command issued from either the client or server lets you prepare to send or receive data using a TCP connection you previously opened. It has the following syntax (colons must be specified as shown):

```
USE devicename:(::mode:terminators)
```

where

devicename	A string of the form |TCP| followed by some number of numeric digits. The numeric portion of the device name is called the device identifier. If the port number is not specified in the **OPEN** parameters, this device identifier must be a unique five-digit TCP port number. If the port number is specified in the **OPEN** parameters (which is the preferred practice), this device identifier can be any unique number, so long as all the TCP device names used by a single job are distinct.
mode	*Optional* — **USE** supports the same mode parameters as **OPEN**. See "OPEN and USE Command Keywords for TCP Devices."
terminators	*Optional* — A list of up to eight user terminator characters that will terminate reads on the TCP binding device. It does not make sense to specify both T mode and user terminators at the same time, but if you do then T mode is ignored.

The simplest form of **USE** takes its mode and terminators parameters from the **OPEN** command, as shown in the following example:

ObjectScript

```
USE "|TCP|4"
```

You can replace, add, or delete mode parameters and user terminators after the device has been opened.

To replace the parameters specified in **OPEN**, specify replacement values in **USE**. In the following example, the **USE** command replaces the **OPEN** mode with PSTE mode and turns off any user terminators:

ObjectScript

```
USE "|TCP|4":(::"PSTE")
```

To add to or delete from the mode parameters specified in **OPEN**, use the "+" sign to introduce mode parameters that will be turned on, and the "-" sign to introduce mode parameters that will be turned off. If you do not specify either "+" or "-", the new set of mode parameters replaces the existing mode parameters. In the following example, the **USE** command turns off Q mode (send immediate) and turns on W mode (wait). It leaves the rest of the mode string unchanged:

ObjectScript

```
USE "|TCP|4":(::"-Q+W")
```

In the following example, the **USE** command leaves the mode string unchanged and specifies a new set of user terminators.

ObjectScript

```
USE "|TCP|4":(::"+":$CHAR(3,4))
```

5.5 READ Command for TCP Devices

Issue the READ command from either the server or the client to read any characters set by either the client or the server.

The syntax is as follows:

```
READ var:timeout
READ *var:timeout
READ var#length:timeout
```

The *timeout* argument, though optional, is strongly recommended because the success or failure of the **READ** is indicated by the value of the $TEST special variable if *timeout* is specified. **$TEST** is set to 1 if the read attempt succeeds before the timeout expires; if the timeout expires, **$TEST** is set to 0.

TCP **READ** timeout is supported for whole seconds or for a fraction of less than a second. TCP **READ** truncates a *timeout* value of 1 second or more to an integer number of seconds (4.9 = 4 seconds). TCP **READ** supports *timeout* values of less than 1 second to the 1/100th of a second (0.9 = nine tenths of a second).

For an SSL connection, a job can wait in the first read or first write command if the other party never issues a read or write command after the connection is established. In this circumstance, Caché supports the read timeout for a **READ** command and write timeout (with /WRITETIMEOUT=n option) for a **WRITE** command. If there is no read or write timeout specified, then the job will wait until the other party issue a read or write command.

You can determine the number of reads performed by the current TCP connection using the **TCPStats()** method of the %SYSTEM.INetInfo class.

5.5.1 READ Modifies $ZA and $ZB

Your application can learn about how the connection and read succeeded by testing the values of **$ZA** and **$ZB**.

5.5.1.1 $ZA and READ Command

$ZA reports the state of the connection. When the 0x1000 bit (4096) is set, this TCP device is functioning in Server mode. When the 0x2000 bit (8192) is set, the device is currently in the Connected state talking to a remote host.

For example, assume that a server-side TCP device is expected to accept a new TCP connection. By looking at **$ZA** and **$TEST** after an initial timed read, the Caché program can distinguish among three cases:

$ZA Value	$TEST Value	Meaning
4096	0	No connection has been accepted.
12288	0	Connection accepted, no data received.
12288	1	Connection accepted and data received.

The following table shows what each bit in **$ZA** represents.

Decimal Value of $ZA	Hexadecimal Value of $ZA	Meaning
2	0x2	Read timed out.
4	0x4	I/O error.
256	0x80	Bad escape sequence received.
4096	0x1000	Server mode.
8192	0x2000	Connected.

5.5.1.2 $ZB and READ Command

$ZB holds the character that terminated the read. This character can be one of the following:

- A termination character, such as a carriage return
- The yth character of a fixed-length `READ x#y`
- The single character of `READ *x`
- An empty string after a timed read expires
- An escape sequence

Note that if a string is terminated with CR LF, then only the CR is placed in **$ZB**.

5.6 WRITE Command for TCP Devices

The WRITE command sends data to a TCP device from the client or the server after you have established connection with **OPEN** and **USE**.

The syntax is as follows:

```
WRITE x
WRITE !
WRITE #
```

5.6.1 How WRITE Works

`WRITE x` sends *x* from the client or server to a buffer after the connection has been established.

`WRITE !` and `WRITE #` do not indicate line and form feed. Instead, they tell Caché to flush any characters that remain in the buffer and send them across the network to the target system.

You can determine the number of writes performed by the current TCP connection using the **TCPStats()** method of the %SYSTEM.INetInfo class.

5.6.2 WRITE Modifies $X and $Y

Caché stores the number of characters in the buffer in the **$X** special variable.

The ASCII characters <return> and <line feed> are not included in this count, as they are not considered part of the record. Flushing the buffer with `WRITE !` resets **$X** to 0, and increases the value of **$Y** by 1. Flushing the buffer with `WRITE #` writes the ASCII character <form feed> as a separate record, and resets **$Y** to 0.

5.6.3 WRITE Command Errors

You can receive a <WRITE> error in any of the following circumstances.

- If you exceed the maximum string size (1024 characters) without flushing the buffer.
- If you flush the write buffer when there are no characters in it (TCP/IP ignores records of 0 length).
- If you send a **WRITE** command from the server to the client before the server receives a connection request from client. (Caché produces the <WRITE> error on the server.)

5.6.4 WRITE Control Commands

The Caché TCP binding device supports a series of control commands with the `WRITE *-n` syntax.

Syntax	Description
`WRITE *-2`	On a server-mode session that is currently connected to a client, this command disconnects from the session. To accept a new session you then execute a new **READ** command on the device.
`WRITE *-3`	Sends any buffered output out the TCP connection; that is, executes a TCP SEND operation on the data in the output buffer. If the data is compressed (/GZIP) stream data, *-3 sends the data without marking the compression endpoint. Resets **$X** to 0. Increments **$Y** by 1. If there is no buffered output, this command does nothing.
`WRITE *-99`	Sends compressed (/GZIP) stream data. First marks the data in the output buffer with a compression endpoint, then sends this compressed stream data by executing a TCP SEND operation on the output buffer data.

5.7 Connection Management

The server maintains only one connection at a time. If a second client tries to connect while another connection is open, TCP/IP places that client in a queue. While in the queue, the second client can write to the port as if it were connected. The data the second client writes remains in a buffer until the first connection is closed and the second client connects.

The second client hangs if it issues a **READ** before the connection exists. Any connection attempt by a third client while the second one is in the queue fails.

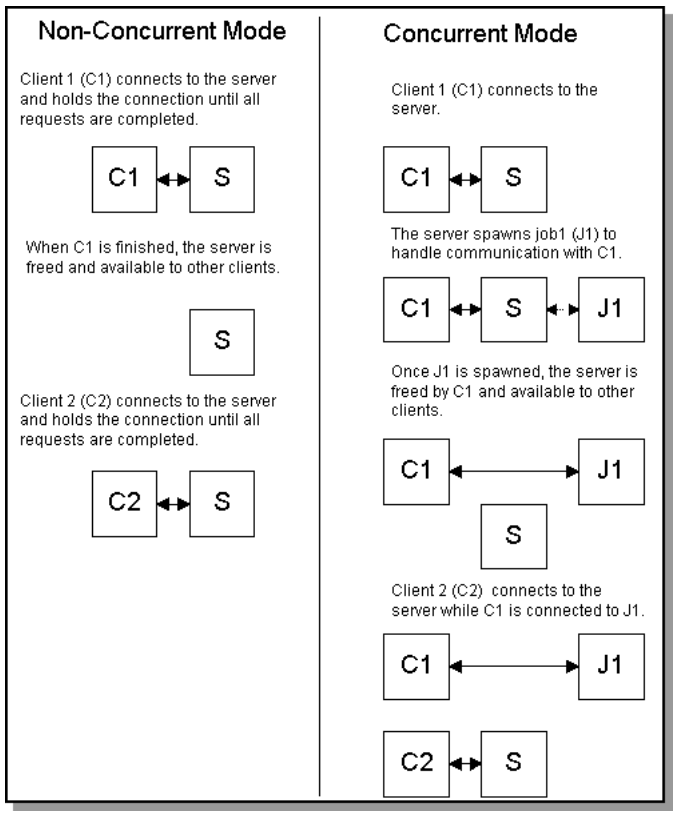
If a client that has already opened a TCP device tries to connect a second time while the first connection still exists, the second **OPEN** command causes a <COMMAND> error. Treating this situation as an error rather than as a **USE** command prevents surprising results. Such unexpected results could occur if an erroneous program thinks it has opened a new connection, when it is actually reusing an existing connection that may have a different destination or different parameters.

To handle multiple clients, see below.

5.7.1 Job Command with TCP Devices

You can use the JOB command to implement a *TCP concurrent server*. A TCP concurrent server allows multiple clients to be served simultaneously. In this mode, a client does not have to wait for the server to finish serving other clients. Instead, each time a client requests the server, it spawns a separate subjob for that client which remains open as long as the client needs it. As soon as this subjob has been spawned (indicated by the return of the **JOB** command), another client may request service and the server will create a subjob for that client as well.

Figure 5–1: Client/Server Connections in the Non-Concurrent and Concurrent Modes.



A concurrent server uses the **JOB** command with the *switch* concurrent server bit (bit 4 or bit 16) set. Bit 16 is the recommended setting.

- If bit 4 is set, the **JOB** command passes to the spawned process the TCP device in the *principal input* and *principal output* process parameters. Whenever you include bit 4 in *switch*, you must specify the TCP device in both *principal input* and *principal output* process parameters. You must use the same device for both *principal input* and *principal output*. Use of bit 4 is not recommended; refer to the JOB command in the *Caché ObjectScript Reference* for further details.
- If bit 16 is set, the **JOB** command passes to the spawned process three separate devices for the TCP device, the *principal input*, and *principal output* process parameters. You specify two of these TCP devices in the **JOB** command, using the *principal input* and *principal output* process parameters. You can also default these parameters, as shown in the following examples: `JOB child:(:16:input:output)` or `JOB child:(:16::))`.

Refer to the JOB command in the *Caché ObjectScript Reference* for further details.

Before you issue the **JOB** command, the device(s) you specify for principal input and principal output must:

- Be open
- Be listening on a TCP port
- Have accepted an incoming connection

After the **JOB** command, the device in the spawning process is still listening on the TCP port, but no longer has an active connection. The application should check **$ZA** after issuing the **JOB** command to make sure that the CONNECTED bit in the state of the TCP device was reset.

The spawned process starts at the designated entry point using the specified TCP device. The TCP device has the same name in the child process as in the parent process. The TCP device has one attached socket. The inherited TCP device is

in S (stream) mode. However, the child process can change the mode with a **USE** command. We recommend that the server open TCP device in the A (accept) mode.

The TCP device in the spawned process is in a connected state: the same state the device would receive after it is opened from a client. The spawned process can use the TCP device with `USE 0` or `USE $P`. It can also use the TCP device implicitly (if *switch*=4). However, for the following reasons *switch*=16 is preferable to *switch*=4:

- When *switch*=4, if a <READ> error occurs on the principal device, the job simply halts, without taking an error trap. This is because when *switch*=4 the TCP device is the principal device. To support error trapping, use *switch*=16 and specify another device for the TCP device.
- When *switch*=4, if the remote TCP device closes down the connection, the job simply halts, without taking an error trap. To override this default behavior and generate a <DSCON> error, you must set the **DisconnectErr()** method of the %SYSTEM.Process class.

You can use the %SYSTEM.Socket class methods, rather than the **JOB** command, to create concurrent TCP server connections. However, note that the %SYSTEM.Socket methods assume that the slave jobs are already started. You can use these methods for concurrent TCP server connections if you do not need the master server to start the slave jobs, and the master server knows Process IDs (PIDs) of the slave jobs.

5.7.2 Job Command Example

The following example shows a very simple concurrent server that spawns off a child job whenever it detects a connection from a client. **JOB** specifies a concurrent server bit *switch* value (value 16) and passes the symbol table (value 1): 16+1=17.

ObjectScript

```
server
  SET io="|TCP|1"
  SET ^serverport=7001
  OPEN io:(:^serverport:"MA"):200
  IF ('$TEST) {
       WRITE !,"Cannot open server port"
       QUIT }
  ELSE { WRITE !,"Server port opened" }
loop
     USE io READ x ; Read for accept
     USE 0 WRITE !,"Accepted connection"
     JOB child:(:17:io:io) ;Concurrent server bit is on
     GOTO loop
child
  WRITE $JOB,! ; Send job id on TCP device to be read by client
  QUIT
client
  SET io="|TCP|2"
  SET host="127.0.0.1"
  OPEN io:(host:^serverport:"M"):200 ;Connect to server
   IF ('$TEST) {
       WRITE !,"cannot open connection" Quit }
   ELSE {
       WRITE !,"Client connection opened"
       USE io READ x#3:200 ;Reads from subjob
       }
   IF ('$TEST) {
       WRITE !,"No message from child"
       CLOSE io
       QUIT }
   ELSE {
       USE 0 WRITE !,"Child is on job ",x
       CLOSE io
       QUIT }
```

The child uses the inherited TCP connection to pass its job ID (in this case assumed to be 3 characters) back to the client, after which the child process exits. The client opens up a connection with the server and reads the child's job ID on the open connection. In this example, the IPv4 format value “127.0.0.1” for the variable *host* indicates a loopback connection to the local host machine. You can set up a client on a different machine from the server if *host* is set to the server's IP

address or name. Further details on IPv4 and IPv6 formats can be found in the section "Use of IPv6 Addressing" in the chapter "Server Configuration Options" in the *Caché Programming Orientation Guide*.

In principle, the child and client can conduct extended communication, and multiple clients can be talking concurrently with their respective children of the server.

Note that this simple example does not contain logic for detecting and handling disconnects or failed read operations.

5.8 Concatenation of Records

In certain situations, TCP concatenates separate records to form a single record. Concatenation can occur if a client or server process issues a series of **WRITE** commands to a TCP port, separated by `WRITE !` or `WRITE #` commands to flush the buffer, whether or not a **READ** command is waiting at the other end of the connection.

The first example below outlines how Process A receives two separate records when it has a **READ** command waiting as Process B writes two records to the TCP port.

```
Process A                                Process B
%SYS> USE "|TCP|41880" R A U 0 W A       %SYS> USE "|TCP|41880" WRITE "ONE",!,"TWO"
<RETURN>                                 <RETURN>
ONE
%SYS> USE 41880 R A U 0 W A
<RETURN>
TWO
```

The second example outlines how Process A receives one concatenated record when it issues its **READ** command after Process B has finished writing two records to the TCP port.

```
Process A                              Process B
.                                      %SYS> USE "|TCP|41880" WRITE "ONE",!,"TWO"
.                                      <RETURN>
ONE
%SYS> USE "/TCP/41880" R A U 0 W A
<RETURN>
ONETWO
```

5.9 Multiplexing Caché TCP Devices

The %SYSTEM.Socket class provides methods for multiplexing Caché TCP devices. The **Fork()** and **Select()** methods allow you to have a single job handling both accepting new connections and reading data from a connected TCP device at the same time. After a listening TCP device received a connection, use **Fork()** to create a new TCP device for reading data. The original listening TCP device continues to accept incoming connections. You use the **Select()** method to wait for both listening and connected TCP devices. When a new connection arrived or data becomes available, **Select()** returns the device name that was signaled.

You can use the **Select()**, **Publish()**, **Export()**, and **Import()** methods to have a master job accept an incoming connection and pass the connected device to a slave job. This slave job could communicate with the remote client.

For further details and program examples, refer to the %SYSTEM.Socket class in the *InterSystems Class Reference*.

5.10 Closing the Connection

Either the client or the server can end a TCP binding connection. The preferred way to close a connection is for the client to issue a CLOSE command for the TCP device. (Alternatively, the client may issue HALT command.) The server should then issue another **READ** command to that device and receive a <READ> error, then issue a CLOSE command for the TCP device.

The reason for this sequence is that, in accordance with the TCP/IP standard, connection resources are maintained for two minutes after a **CLOSE**, but only for the "active closer" — the process that performs the **CLOSE** first. Thus it is preferable to close the client first, because the resources of the server are usually more limited than those of the clients.

5.10.1 Disconnect with CLOSE Command

Issue this form of the **CLOSE** command from the client or server:

```
CLOSE "|TCP|devicenum"
```

As stated above, it is preferable for the client to issue the **CLOSE** command first. If the server issues the **CLOSE** command first, the client gets a <WRITE> error and should then issue a **CLOSE** command.

5.10.1.1 JOBSERVER Resources

If you are writing a Caché server to interface with clients over which you have no control, the server process must issue the **CLOSE** to close the TCP connection. The **CLOSE** command does close the connection as far as Caché is concerned, but internally TCP/IP retains resources for this connection on the server for up to two minutes.

This can have unexpected results when JOBSERVERs are used to service TCP/IP jobs. When a JOBSERVER process performs a halt, the process immediately returns to the pool of available JOBSERVER processes, but its resources are retained internally for up to two minutes. Because JOBSERVER processes are assigned on a first-available basis, it is possible for a heavy load from a relatively small number of clients to exhaust the resources of a JOBSERVER process.

To avoid this problem, a TCP/IP server opened by a **JOB** running under a JOBSERVER should explicitly issue a **CLOSE**, and then issue a brief **HANG** before the final **QUIT** (or **HALT**) command. In accordance with TCP/IP specification, a **HANG 120** is required to guarantee no resources remain in use between incarnations in JOBSERVER. In practice, a **HANG** of one second is usually sufficient to evenly distribute resource load among JOBSERVER processes.

5.10.2 Server Disconnects with WRITE *-2 Command

Use the `WRITE *-2` command when the server needs to disconnect from one client so that it can accept a connection from another client:

ObjectScript

```
WRITE *-2
```

This option replaces the **USE** command with the "DISCONNECT" parameter, which is still supported for compatibility with older versions, but will be eliminated in a future release.

To accept a new session, you then execute a new **READ** command on the device.

The client that was disconnected should issue a **CLOSE** command to the TCP device to free up that device.

5.10.3 Automatic Disconnection

The TCP binding connection closes automatically under these conditions:

- A Caché fatal error
- RESJOB of the client or server process
- ccontrol stop
- ccontrol force

5.10.4 Effects of Disconnection

The effect of a disconnection on data remaining in the output buffer is determined by the /CLOSEFLUSH setting established during **OPEN** or **USE**. The default is to flush the data.

If one side closes a connection but the other side issues new **WRITE** commands, the first of these **WRITE** commands may succeed. Any additional **WRITE** commands receive a <WRITE> error.

From the client side, all **READ** commands to the side that closed the connection receive <READ> errors. The device must be closed and reopened to reestablish communication with the server.

From the server side, the first **READ** after a <READ> or <WRITE> error waits for and accepts a new connection.

You can use the **%SYSTEM.TCPDevice.GetDisconnectCode()** method to return the internal error that resulted in a <READ> or <WRITE> error on the current TCP device. $IO must be a TCP device.

6

UDP Client/Server Communication

This chapter describes how to set up remote communication between processes using UDP. For local communication between processes using pipes or using Interjob Communication (IJC) devices, refer to the Local Interprocess Communication chapter of this manual.

Caché supports two Internet Protocols (IP): TCP and UDP. These Internet Protocol allow Caché processes to communicate with processes on local or remote systems, whether or not those processes are running Caché.

- TCP: the Caché Transmission Control Protocol (TCP) binding. Establishes a two-way connection between a server and a single client. Provides reliable byte stream transmission of data with error checking and correction, and message acknowledgement. For details, refer to the TCP Client/Server Communication chapter of this manual.
- UDP: the Caché User Datagram Protocol (UDP) binding. Provides two-way message transfer between a server and a large number of clients. UDP is not connection-based; each data packet transmission is an independent event. Provides fast and lightweight data transmission for local packet broadcasts and remote multicasting. Inherently less reliable than TCP. Does not provide message acknowledgement.

UDP is supported through the %Net.UDP class. This class provides methods to **Send()** a packet to a specified destination and port, to **Recv()** a packet from the socket, and to **Reply()** to the transmitter of the last received packet.

The destination is identified as a local host name or an IPv4 or IPv6 host address. The port can be either a specified port number or a dynamic port assignment.

6.1 Establishing a UDP Socket

To use UDP, you must use the **%New()** method to create a UDP socket object. This object instance is then used to send, receive, and reply to packet transmissions.

When you create a UDP socket object you can specify the port number and the host address, as shown in the following example:

ObjectScript

```
SET UPDOref=##class(%Net.UDP).%New(3001,"0.0.0.0")
```

Both the port number and the host address are optional. The **%New()** method returns the oref (object reference) of the UDP socket object instance.

There are two sides to a UDP transmission:

- The server waits to receive a request and then provides the requested information. Thus this side of the transmission may be referred to as the Receiver or the Provider. When a provider creates an UDP object, it must define a port number on which it will receive requests.
- The client sends a request for information and then receives a reply. Thus this side of the transmission may be referred to as the Sender or the Requestor. When a requestor creates an UDP object, it can use a dynamic port number. The default is 0. When it sends a packet, it must specify the host name and port number of the provider.

6.2 The Host Address

The **Send()** method specifies the binary address of the destination. This is a binary version of the host address. You must create this binary host address by using the **GetHostAddr()** method, as follows:

ObjectScript

```
SET client=##class(%Net.UDP).%New()
SET addrbin=##class(%Net.UDP).GetHostAddr("172.16.61.196")
WRITE client.Send("message text",addrbin,3001)
```

You can specify a host name, an IPv4 address, or an IPv6 address to **GetHostAddr()**, as shown in the following examples:

ObjectScript

```
SET hostname="MYLAPTOP"
SET IPv4="172.16.61.196"
SET IPv6="::1"
SET flag=$SYSTEM.INetInfo.CheckAddressExist(hostname)
IF flag=1 { SET addrbin=##class(%Net.UDP).GetHostAddr(hostname)
            WRITE "host name valid",! }
   ELSE { WRITE "not a hostname: ",hostname,! }
SET flag=$SYSTEM.INetInfo.CheckAddressExist(IPv4)
IF flag=1 { SET addrbin=##class(%Net.UDP).GetHostAddr(IPv4)
            WRITE "IPv4 valid",! }
   ELSE { WRITE "not an IPv4 address: ",IPv4,! }
SET flag=$SYSTEM.INetInfo.CheckAddressExist(IPv6)
IF flag=1 { SET addrbin=##class(%Net.UDP).GetHostAddr(IPv6)
            WRITE "IPv6 valid",! }
   ELSE { WRITE "not an IPv6 address: ",IPv6,! }
```

You can expand this binary host address back to the host name using the **AddrToHostName()** method, as shown in the following example:

ObjectScript

```
SET addrbin=##class(%Net.UDP).GetHostAddr("MYLAPTOP")
WRITE $SYSTEM.INetInfo.AddrToHostName(addrbin)
```

You can use the **LocalHostName()** method to determine your host name. You can use the **HostNameToAddr()** method to translate a host name to an IPv4 or IPv6 address, as shown in the following example:

ObjectScript

```
SET localhost=$SYSTEM.INetInfo.LocalHostName()            /* get host name */
   WRITE "local host name is ",localhost,!
SET addrbin=##class(%Net.UDP).GetHostAddr(localhost)      /* compress to binary address */
   WRITE "binary form of IP address is ",addrbin,!
SET hostname=$SYSTEM.INetInfo.AddrToHostName(addrbin)     /* expand binary address to host name */
   WRITE "binary IP address expands to ",hostname,!
SET ipaddr=$SYSTEM.INetInfo.HostNameToAddr(hostname)      /* host name to IP address */
   WRITE "hostname corresponds to IP address ",ipaddr,!
```

6.2.1 IPv4 and IPv6

UDP supports both IPv4 and IPv6 Internet protocols. Because these protocols are incompatible, both the server and the client must use the same Internet protocol or the transmission will fail.

An IPv4 address has the following format. *n* is a decimal integer in the range 0 through 255:

```
n.n.n.n
```

You can specify the IPv4 protocol as `"0.0.0.0"`.

An IPv6 address has the following full format. *h* is a hexadecimal number with four hexadecimal digits:

```
h:h:h:h:h:h:h:h
```

Commonly, IPv6 addresses are abbreviated by eliminating leading zeros and replacing consecutive sections of zeros with a double colon (::); only one double colon may be used in an IPv6 address. By using IPv4 abbreviation rules, you can specify the IPv6 protocol as `"::"` (meaning that all eight *h* sections have the value 0000).

To establish the Internet protocol:

- The client must establish either IPv4 or IPv6 in the %New() method. The default is IPv4.
- This must match the IPv4 or IPv6 protocol specified in the **GetHostAddr()** method and supplied (in binary form) in the **Send()** method.

The following is an IPv4 transmission:

ObjectScript

```
Server
 SET sobj=##class(%Net.UDP).%New(3001,"127.0.0.1")


  SET inmsg=sobj.Recv()
```

ObjectScript

```
Client
 SET cobj=##class(%Net.UDP).%New()   /* the default is IPv4 */
 SET bhost=##class(%Net.UDP).GetHostAddr("127.0.0.1")
 SET outmsg="this is the message to send"
 WRITE cobj.Send(outmsg,bhost,3001)
```

The following is an IPv6 transmission:

ObjectScript

```
Server
 SET x=##class(%SYSTEM.INetInfo).IsIPV6Enabled()
 IF x=1 {
    SET sobj=##class(%Net.UDP).%New(3001,"::1")


  SET inmsg=sobj.Recv() }
  ELSE {WRITE "IPv6 not enabled" }
```

ObjectScript

```
Client
 SET cobj=##class(%Net.UDP).%New(0,"::")
 SET bhost=##class(%Net.UDP).GetHostAddr("::1")
 SET outmsg="this is the message to send"
 WRITE cobj.Send(outmsg,bhost,3001)
```

Methods for handling host addresses are found in the %SYSTEM.INetInfo class documentation. Further details on IPv4 and IPv6 formats can be found in the section "Use of IPv6 Addressing" in the chapter "Server Configuration Options" in the *Caché Programming Orientation Guide*.

7

Sequential File I/O

This chapter describes using sequential files in Caché. All operating systems consider disk I/O files as sequential files. Windows systems consider printers as sequential file I/O devices (unless the printer is connected through a serial communications port). UNIX® systems consider printers as terminal I/O devices. For further details on printers, refer to the Printers chapter of this manual.

7.1 Using Sequential Files

This section discusses how Caché processes sequential files. It provides an introduction to sequential file I/O and descriptions of the relevant commands.

- To gain access to a sequential file, you must first open the file using the **OPEN** command, supplying the name of the file as an argument. You also, optionally, specify **OPEN** mode parameters. If the **OPEN** specifies a file that does not exist, a mode parameter specifies whether or not to create a new file. You can open multiple files concurrently.
- After opening a sequential file, you must specify a **USE** command to access the file, supplying the name of the file as an argument. The **USE** command makes the specified file the current device; therefore you can only use one file at a time. The **USE** command can also specify mode parameters.
- You then can issue multiple **READ** or **WRITE** commands against the file. Each **READ** or **WRITE** command delivers one record to or from the file. You cannot write to the file unless it has been opened with the "W" mode parameter. Attempting to read past the end of the file causes an <ENDOFFILE> error.
- You can use the **$ZSEEK** function to set the file position, specified by character count offset from the beginning, current position, or end of the sequential file. The **$ZPOS** special variable contains the current character count position from the beginning of the current sequential file.
- Once you have completed file I/O, you issue a **CLOSE** command to close the sequential file.

These operations can also be performed using the methods of the %Library.File class.

The **%Library.File.Exists()** method tells you whether a sequential file with the specified name already exists.

The *%Library.File.Size* property returns the number of characters currently in the sequential file.

The *%Library.File.DateModified* property is updated with the current local date and time when a file is opened, and —if it has been modified—when it is closed.

The *%Library.File.IsOpen* property only returns 1 if the file has been opened by the **%Library.File.Open()** method; the **OPEN** command does not set this boolean property.

7.1.1 Specifying a File

A sequential file can be specified by a canonical (full) pathname or a relative (partial) pathname that the system expands to a full pathname. A pathname can be canonical (c:\InterSystems\Cache\mgr\user\myfiles\testfile.txt) or relative to the current directory (testfile.txt). A leading period (.) specifies the current directory. A leading double period (..) specifies the parent of the current directory. If the **OPEN** command creates a new file, the specified directory must already exist.

The following Windows examples all create a file in the current namespace (USER) directory:

- full pathname: `OPEN "C:\InterSystems\Cache\mgr\user\testfile1.txt":("WNS"):10`
- filename expansion: `OPEN "testfile2.txt":("WNS"):10`
- current directory expansion: `OPEN ".\testfile3.txt":("WNS"):10`

The following Windows example creates a file in an existing child directory of the current namespace (USER) directory:

- child of current directory: `OPEN "mytemp\testfile4.txt":("WNS"):10`

The following Windows examples create a file using parent directory (..) syntax:

- parent directory (`C:\InterSystems\Cache\mgr\`): `OPEN "..\testfile5.txt":("WNS"):10`
- current directory (child of parent directory) `C:\InterSystems\Cache\mgr\user\`: `OPEN "..\user\testfile6.txt":("WNS"):10.`
- another child of parent directory `C:\InterSystems\Cache\mgr\temp\`: `OPEN "..\temp\testfile7.txt":("WNS"):10.`
- parent of parent directory `C:\InterSystems\Cache\`: `OPEN "..\..\testfile8.txt":("WNS"):10.`

Windows pathnames use a \ (backslash) directory separator; UNIX pathnames use a / (slash) directory separator. Valid characters may be 8-bit ASCII, or ISO Latin-1 Unicode.

A Windows file pathname specification has the following format:

```
device:\directory\file.type
```

For example, `C:\InterSystems\Cache\mgr\user\myfiles\testfile.txt`. The *type* suffix is optional.

A UNIX® file pathname specification has the following format:

```
/directory/name
```

A file pathname must not exceed 256 characters when fully expanded. If the pathname length of all directories exceeds 256, a <DIRECTORY> error is generated. If the pathname length exceeds 256 because of the length of the filename, a <NAMEADD> error is generated.

A UNIX® file pathname can include up to 255 characters of any type. While the characters period (“.”) and underscore (“_”) can appear anywhere in the filename, you typically use them to divide the name into meaningful parts. For example, you can define a filename pat_rec.dat, using .dat as the file type.

When accessing files in the current UNIX® default directory, you usually need to specify only the name. The system fills in default values for the directory.

A DLL name can be specified as a full pathname, or a partial pathname. If you specify a partial pathname, Caché expands it to the current directory. Generally, DLLs are stored in the binary directory ("bin"). To locate the binary directory, call the **BinaryDirectory()** method of the %SYSTEM.Util class.

7.1.1.1 File Pathname Tools

If the current device is a sequential file, $ZIO contains the full pathname of that file.

You can use $ZSEARCH to return the full file specification (pathname and filename) of a specified file or directory. The filename may contain wild cards that **$ZSEARCH** uses to return a series of fully qualified pathnames that satisfy the wild carding.

The %Library.File class contains numerous methods that provide file system services. These include:

- **NormalizeDirectory()**, which returns the full pathname of a specified file or directory.
- **NormalizeFilenameWithSpaces()**, which handles spaces in pathnames as appropriate for the host platform. If a pathname contains a space character, pathname handling is platform-dependent. Windows and UNIX® permit space characters in pathnames, but the entire pathname containing spaces must be enclosed in an additional set of double quote (") characters. This is in accordance with the Windows `cmd /c` statement. For further details, specify `cmd /?` at the Windows command prompt.

7.1.1.2 Tilde (~) Expansion

In Windows pathnames, a tilde (~) indicates 8.3 compression of long names. For example: c:\PROGRA~1\. To convert compressed directory names, use the **NormalizeDirectory()** method of the %Library.File class.

In UNIX® pathnames, you can use tilde (~) expansion to indicate the current user's home directory or the home directory of a specified user:

- ~ and ~/myfile.txt are expanded to the current user's home directory: `/Users/techwriter/` and `/Users/techwriter/myfile.txt`, respectively.
- ~guest/myfile.txt is expanded to the home directory of user "guest": `/Users/guest/myfile.txt`. However, if user "guest" does not exist, Caché expands to the current user's full directory pathname and appends `~guest/myfile.txt` as a literal: `/Users/techwriter/Cache/mgr/user/~guest/myfile.txt`.
- ~myfile.txt and ~123.txt are appended to the current user's full directory pathname as a literal: `/Users/techwriter/Cache/mgr/user/~myfile.txt` and `/Users/techwriter/Cache/mgr/user/~123.txt`, respectively.

7.1.2 OPEN Command

OPEN opens a Caché sequential file. Remember that you cannot use the **OPEN** command to open a Caché database file.

The **OPEN** command by itself does not prevent another process from opening the same sequential file. You can govern concurrent sequential file access by using the **OPEN** command "L" mode parameter and/or the ObjectScript **LOCK** command. File locking support is provided by the file access rules of the underlying operating system.

Caché allocates each process' open file quota between database files and files opened with the ObjectScript **OPEN** command. When an **OPEN** command causes too many files to be allocated to **OPEN** commands, a <TOOMANYFILES> error occurs. The Caché maximum number of open files for a process is 1,024. The actual maximum number of open files for each process is a platform-specific setting. For example, Windows defaults to a maximum of 998 open files per process. Consult the operating system documentation for your system.

7.1.2.1 OPEN Syntax

```
OPEN filename{{:({parameters{:reclength{:terminators}}})}{:timeout}}
```

where

Argument	Description
filename	Any valid file specification, enclosed in quotation marks. Valid characters may be 8-bit ASCII, or ISO Latin-1 Unicode.
parameters	*Optional* — A string of one-letter codes, enclosed in quotation marks, that define the file format and types of operations you can perform. (You may also specify *parameters* using keywords that begin with the slash (/) character.) See the table "OPEN Mode Parameters," for definitions of these codes. If the parameters do not include R or W, then R is the default. This system-wide default open mode can be configured by setting the *OpenMode* property of the Config.Miscellaneous class. To open a new file, you must specify the parameter N for new. Otherwise, the **OPEN** will hang or return unsuccessfully from a timeout. If the parameters do not include S, V, F, or U, then the default for a new Windows or UNIX® file is S, and the default for an existing file is the mode specified when the file was created. If A is not specified, **WRITE** operations will overwrite the previous contents of the file. Parameters are applied in left-to-right order.
reclen	*Optional* — For Windows and UNIX systems, specifies the maximum record length for (S) and (U) records, or the absolute record length for fixed-length (F) records. Ignored for variable-length (V) records. Default value is 32767.
terminators	*Optional* — A string of user-defined record terminators that apply to stream mode only. They let you override the default terminators: carriage return, line feed, and form feed. User-defined terminators only apply to input, they do not affect how data is written to the file (terminators are written to a file as special characters). If there's more than one user-defined terminator, it is treated as a list of terminators, not a multi-character sequence to be used as a single terminator.
timeout	*Optional* — Number of seconds during which Caché attempts to open the file. If it does not succeed within this interval, it sets **$TEST** to 0 and returns control to the process. If it succeeds, it sets **$TEST** to 1.

The *timeout* argument, though optional, is strongly recommended because the success or failure of **OPEN** is indicated by the value of the $TEST special variable, and **$TEST** is only set if *timeout* is specified. **$TEST** is set to 1 if the open attempt succeeds before the timeout expires; if the timeout expires, **$TEST** is set to 0.

7.1.2.2 OPEN Mode Parameters

You can specify the **OPEN** mode parameters in either of two ways:

- A letter code string, such as "VRWN", enclosed in quote characters. Each letter specifies a parameter. Letter codes may be specified in any order; because Caché executes them in left-to-right order, interactions between letter codes may dictate a preferred order in some cases.
- A series of /*keyword* parameters, not quoted. These parameters are separated by colons. Keyword parameters may be specified in any order; because Caché executes them in left-to-right order, interactions between parameters may dictate a preferred order in some cases.

When specifying a combination of letter code parameters and keyword parameters, specify the letter code string first, followed by the keyword parameter(s), separated with colons. The following example specifies three letter code parameters, followed by two keyword parameters, followed by the *reclen* and *timeout* arguments.

ObjectScript

```
OPEN "mytest":("WNS":/OBUFSIZE=65536:/GZIP=0:32767):10
```

Table 7–1: OPEN Mode Parameters

Letter Code	Keyword	Description
N	/NEW	New file. If the specified file does not exist, the system creates the file. If the specified file already exists as a ReadOnly file, the system deletes the old file and replaces it with a new one with the same name (permissions permitting). Note that file locking should be used to prevent concurrent processes using this parameter overwriting the same file. If the "N" mode is not specified and the file specified in **OPEN** does not exist, the Windows and UNIX® default is to not create a new file. This behavior is configurable using the **FileMode()** method of the %SYSTEM.Process class. The system-wide default behavior can be established by setting the *FileMode* property of the Config.Miscellaneous class.
E	/CREATE *or* /CRE	Create a file if it does not exist. Does not delete and recreate an existing file, as the "N" mode does. The default is to not create a new file. This default is overridden if the **FileMode()** method of the %SYSTEM.Process class, or the *FileMode* property of the Config.Miscellaneous class is enabled.
D	/DELETE[=*n*] *or* /DEL[=*n*]	Delete File: Specifies that the file should be automatically deleted when it is closed. /DELETE or /DELETE=*n* for nonzero values of *n* enable the parameter code. /DELETE=*n* for a zero value of *n* disables the parameter code. The default is to not delete a file.
R	/READ	Read: Caché permits **READ** access the file. Other processes may also access this file (however, see "L" parameter). If you attempt to open a nonexistent file in "R" mode, the process hangs. To prevent this situation, use a timeout. "R" is the default for all platforms. The system-wide default open mode can be configured by setting the *OpenMode* property of the Config.Miscellaneous class.
W	/WRITE *or* /WRI	Write: Caché permits **WRITE** access to the file. In Windows and UNIX®, "W" gives the process shared write access to the file, with exclusive write access to the record. Use "WL" to specify exclusive write access to the file. If you attempt to open a nonexistent file in "W" mode, the process hangs until the file is created or the process is resolved by a timeout, a Process Terminate, or RESJOB. "R" is the default for all platforms. The system-wide default open mode can be configured by setting the *OpenMode* property of the Config.Miscellaneous class. Can be used with /OBUFSIZE.
L		Locked Exclusive: Use this mode with the "W" (Write) mode to specify exclusive write access to a file. "WL" or "WRL" specifies that the current process has exclusive write access to the file. A file opened with "RL" may still have shared read access. The effects of the "L" mode on concurrent opens are different in Windows and UNIX®. Refer to the "OPEN Mode Locking" section, below, for further details. On UNIX® systems if one process specifies "WL" (or "WRL") access to a file, other processes requesting read access to that file must specify "RL" so that UNIX® can coordinate file locking.

Letter Code	Keyword	Description
A	/APPEND *or* /APP	Append: **WRITE** operations append data to the end of an existing file. The default is to overwrite existing data, rather than append.
S	/STREAM	Stream format with carriage return, line feed, or form feed as default terminators. Jobbed processes that inherit TCP devices are automatically set to “S” format. You can reset the format with the **USE** command. S, V, F, and U modes are mutually exclusive. Stream record format is the default.
V	/VARIABLE	Variable length: Each **WRITE** creates one record. For Windows and UNIX®, a variable record can be of any length; the *reclen* argument is ignored. Do not attempt to insert records at any point other than the end of a variable-length sequential file; a **WRITE** will render inaccessible all data in the file from the point after the **WRITE** on. S, V, F, and U modes are mutually exclusive. Stream record (S) format is the default. A variable-length record written using a translation table, such as Unicode data using UTF8 translation, can result in a stored record with a different string length than the input data. Caché uses the original input string length when reading this record.

F	/FIXED
	or
	/FIX
		Fixed length: Each record is the length specified in the *reclen* argument. For example:
		ObjectScript
		`OPEN "myfile":("RF":4)`
		`USE "myfile":0`
		`READ x:5`
		This example reads the first 4–character record into the variable x. This works only for **READ** operations (not **WRITE** operations). S, V, F, and U modes are mutually exclusive.

U	/UNDEFINED	Undefined length: Specifies that file records have an undefined length and therefore **READ** operations must specify the number of characters to read. The maximum record length is specified in the *reclen* argument. No translation on output. Default value is the maximum string length. S, V, F, and U modes are mutually exclusive.

Letter Code	Keyword	Description
K\name\ *or* Knum	/TRANSLATE[=*n*]: /IOTABLE[=*name*] *or* /TRA[=*n*]: /IOT[=*name*]	I/O Translation Mode: When you use the "K" parameter for a device, I/O translation will occur for that device if translation has been enabled system-wide. You identify the previously defined table on which the translation is based by specifying the table's name. When using keywords you specify /TRANSLATE to enable I/O translation (*n*=1 to enable; *n*=0 to disable), and /IOTABLE=*name* to specify the translation table to use. For a list of available translation tables, refer to "Encoded Translation" in the **$ZCONVERT** function documentation. The + and - options for turning protocols on and off are not available with the K protocol. (The older form Knum, where "num" represents the number of the slot the table is loaded in, is being phased out but is still supported. The system manager can display slot numbers in the %NLS utility in the selection window for each table type.) This parameter may be used with either the **OPEN** command or the **USE** command.
Y\name\ *or* Ynum	/XYTABLE[=*name*] *or* /XYT[=*name*]	$X/$Y Action Mode: When you use the "Y" parameter for a device, the system uses the named $X/$Y Action Table. You identify the previously defined $X/$Y Action Table on which translation is based by specifying the table's name. $X/$Y action is always enabled. If "Y" is not specified and a system default $X/$Y is not defined, a built in $X/$Y action table is used. The + and - options for turning protocols on and off are not available with the Y protocol. (The older form Ynum, where "num" represents the number of the slot the table is loaded in, is being phased out but is still supported. The system manager can display slot numbers in the NLS utility in the selection window for each table type.) This parameter may be used with either the **OPEN** command or the **USE** command.
	/NOXY [=*n*]	No $X and $Y processing: /NOXY or /NOXY=*n* (for nonzero values of *n*) disables $X and $Y processing. This can substantially improve performance of READ and WRITE operations. The values of the $X and $Y variables are indeterminate, and margin processing (which depends on $X) is disabled. /NOXY=0 enables $X and $Y processing; this is the default. This parameter may be used with either the **OPEN** command or the **USE** command.
	/OBUFSIZE=*int*	Output Buffering: Creates an output WRITE buffer. The *int* variable is an integer that specifies the size of the buffer in bytes. May only be used when the file is open for write only ("W", not "R" or "RW"). May provide significant performance improvement when performing multiple small writes, especially over a WAN. However, data in buffer may be lost if a system crash occurs. Data in buffer is flushed to disk upon **CLOSE**, or **WRITE *-1** or **WRITE *-3**.

Letter Code	Keyword	Description
	/GZIP [=*n*]	GZIP Compression: Specifies GZIP-compatible stream data compression. /GZIP or /GZIP=*n* (for nonzero values of *n*) enables compression on WRITE and decompression on READ. /GZIP=0 disables compression and decompression. Before issuing /GZIP=0 to disable compression and decompression, check the $ZEOS special variable to make sure that a stream data read is not in progress. /GZIP compression has no effect on I/O translation, such as translation established using /IOTABLE. This is because compression is applied after all other translation (except encryption) and decompression is applied before all other translation (except encryption).

7.1.2.3 OPEN Argument Keywords

The following table describes the **OPEN** command argument keywords for sequential files:

Table 7–2: OPEN Keyword Arguments for Sequential Files

Keyword	Default	Description
/PARAMS=*str* *or* /PAR=*str*	No default	Corresponds to the *parameters* positional parameter. (It provides a way to specify a parameter letter code string in a position-independent way).
/RECORDSIZE=*int* *or* /REC=*int*	No default	Corresponds to the *reclen* positional parameter, which establishes a record size for fixed-length records. (Currently only implemented for **READ** operations.)
/TERMINATOR=*str* *or* /TER=*str*	No default	Corresponds to the *terminators* positional parameter, which establishes user-defined terminators. *str* is a string of user-defined record terminators that apply to stream mode only. They let you override the default terminators: carriage return, line feed, and form feed. User-defined terminators only apply to input, they do not affect how data is written to the file (terminators are written to a file as special characters). If there's more than one user-defined terminator, it is treated as a list of terminators, not a multi-character sequence to be used as a single terminator.

7.1.2.4 OPEN Mode Locking

When two processes attempt to open the same sequential file, the second **OPEN** succeeds or fails based on the mode used by the first **OPEN**. The following tables show the interactions between two opens using exclusive (“L”) and non-exclusive read and write modes. Note that the interpretation of these interactions is platform-dependent. Tables are provided for Windows operating systems and UNIX® operating systems.

In the following tables, the horizontal axes indicates the open mode of the first **OPEN** and the vertical axis indicates the open mode of the second **OPEN**. A 1 indicates that the second **OPEN** succeeds; a 0 indicates that the second **OPEN** fails.

Table 7–3: Windows OPEN Mode Interactions

	W	RW	RL	WL	RWL	R
W	1	1	1	0	0	1
RW	1	1	1	0	0	1
RL	1	1	1	0	0	1
WL	0	0	0	0	0	0
RWL	0	0	0	0	0	0
R	1	1	1	0	0	1

For Windows systems, the interactions in this table apply equally to concurrent opens from the same Caché instance, concurrent opens from two different Caché instances, or concurrent opens by Caché and a non-Caché application (with restrictions on non-Caché applications, as described below).

Table 7–4: UNIX® OPEN Mode Interactions

	W	RW	RL	WL	RWL	R
W	1	1	1	1	1	1
RW	1	1	1	1	1	1
RL	1	1	1	0	0	1
WL	1	1	0	0	0	1
RWL	1	1	0	0	0	1
R	1	1	1	1	1	1

For UNIX® systems, the interactions in this table only to concurrent opens from the same Caché instance. They do not govern concurrent opens from two different Caché instances, or concurrent opens by Caché and a non-Caché application.

Interactions with Non-Caché Software

On Windows systems, opening a sequential file in Caché for “WL” write access generally prevents a non-Caché application from opening the sequential file for write access. Similarly, a non-Caché application opening a sequential file for write access generally prevents a Caché process from concurrent “WL” write access.

However, certain non-Caché applications, including the Notepad and WordPad applications, open a file, make a copy of the file in shared mode, and then immediately close it. Thus a Caché process could still open the file in “WL” mode. An error would occur when one of these non-Caché applications then either attempts to save changes from their copy to the original, or attempts to reopen the original file. A more serious situation can occur as follows: If one of these non-Caché applications opens a file, then Caché opens, modifies, and closes the file, then the non-Caché application saves changes to the file, the changes made by both processes are saved, and the integrity of the file data could be compromised.

On UNIX® systems, opening a sequential file in Caché for “WL” write access generally has no effect on the behavior of non-Caché applications. You must use locks to reliably restrict write access from non-Caché applications.

7.1.2.5 Examples

The following example opens the file "LUDWIG.B" in the current directory. Because it specifies no mode parameters, it opens the file with read access and in stream mode by default:

ObjectScript

```
OPEN "LUDWIG.B"
```

The following example opens a new file "LIST.FILE" in the current directory, with write access, in stream format. Notice that you do not need parentheses when you include only the first of the arguments they would normally enclose.

ObjectScript

```
OPEN "LIST.FILE":"WNS"
```

The following example opens a file "CARDS" in the current directory, with read and write access, and 80-character fixed-length records.

ObjectScript

```
OPEN "CARDS":("FRW":80)
```

The following example opens the stream-format file "STRNG" in the directory c:\usr\dir, with non-default terminators.

ObjectScript

```
OPEN "c:\usr\dir\STRNG":("S"::$CHAR(0)_$CHAR(255))
```

7.1.3 USE Command

The **USE** command makes an opened sequential file the current device. You can have many open sequential files, but you can *use* only one sequential file at a time.

7.1.3.1 Syntax

```
USE file:position
```

where

Table 7–5: USE Command Parameters

Argument	Description
file	Any valid file specification, enclosed in quotation marks. The specified file must already have been opened.
position	*Optional* — The position of the next **READ** or **WRITE** within the file. The *position* value is a numerical expression whose meaning depends on the record format of the file. For fixed-length records, *position* is the absolute record number, relative to zero, where each record contains the number of characters specified in the previous **OPEN** command. For stream or variable-length records, *position* is the absolute byte position relative to zero. The default is to read or write records sequentially from the beginning of the file. You can use the $ZSEEK function to set the file position, specified by character count offset from the beginning, current position, or end of the sequential file. The $ZPOS special variable contains the current character count position from the beginning of the sequential file.

7.1.3.2 USE-Only Command Keywords

In addition to the command keywords that it shares with **OPEN**, listed above, the **USE** command has its own set of keywords:

Table 7–6: USE-Only Command Keywords for Sequential Files

Keyword	Default	Description
/POSITION=*n*	Current file position. (The file pointer position is at the beginning of a file when it is first opened, unless the file was opened in append mode. In that case, the file pointer position is at the end of the file.)	Corresponds to the positional parameter, which sets the position of the next **READ** or **WRITE** within a file.

7.1.4 READ and WRITE Commands

After a positioned **READ** or **WRITE**, subsequent **READ** or **WRITE** operations proceed sequentially until the next **USE** command with a position parameter.

7.1.4.1 READ Command

The **READ** command reads data from the current device, one record at a time. Reading past the end of file causes an <ENDOFFILE> error.

Syntax

```
READ x#n:timeout
```

where

Argument	Description
x	The variable that will hold the record read from the file.
n	*Optional* — For a variable-length read, the number of characters to read, specified as an integer. For a fixed-length read, this argument is ignored.
timeout	*Optional* — The number of seconds to wait for the read operation to complete before timing out. Either an integer value or a variable that resolves to an integer.

The *timeout* argument, though optional, is strongly recommended because the success or failure of the **READ** is indicated by the value of the $TEST special variable if *timeout* is specified. **$TEST** is set to 1 if the read attempt succeeds before the timeout expires; if the timeout expires, **$TEST** is set to 0.

The following example shows a **READ** operation reading fixed-length records from a Windows sequential file. It creates a sequential file, writes data into it, then closes the file. It then opens this file for fixed-length reads of 4 characters (`"RF":4`). It sets the **USE** position argument to the first record (record 0); each read operation advances this position. A FOR loop reads each four-character record into a subscripted variable. The **ZWRITE** command then displays all of these subscripted local variables and their values.

ObjectScript

```
SET myf="C:\InterSystems\Cache\mgr\temp\myfixedlengthfile"
OPEN myf:("NW") USE myf WRITE "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
CLOSE myf
OPEN myf:("RF":4) USE myf:0 FOR i=1:1:7 {READ x(i):5}
CLOSE myf
ZWRITE
```

7.1.4.2 WRITE Command

The **WRITE** command writes data, one record at a time, to the sequential file that is the current device.

Syntax

```
WRITE x
```

where

Argument	Description
x	The data in variable x is written as one record in the sequential file.

7.1.4.3 Example

The following example reads the third, fourth, and fifth records of a fixed-length file:

ObjectScript

```
SET myfile="FIXED.LEN"
OPEN myfile:("FR":100)
USE myfile:2
READ var1(3),var1(4),var1(5)
```

7.1.5 CLOSE Command

The **CLOSE** command relinquishes ownership of a sequential file.

If the specified file is not open or does not exist, Caché ignores **CLOSE** and returns without issuing an error.

7.1.5.1 Syntax

```
CLOSE file
CLOSE file:"D"
CLOSE file:("R":newname)
```

Argument	Description
file	Any valid file specification, enclosed in quotation marks. The specified file must already have been opened. In UNIX pathnames, you can use tilde (~) expansion to indicate the current user's home directory. For example: ~myfile or ~/myfile.
"D"	Closes and deletes the file with the name specified in the argument.
("R":*newname*)	Closes the file with the name specified in the argument and renames it *newname*.

7.1.5.2 CLOSE-Only Command Keywords

The following table describes the keywords for controlling sequential files with only the **CLOSE** command.

Table 7–7: CLOSE-Only Command Keywords for Sequential Files

Keyword	Default	Description
/DELETE[=*n*] *or* /DEL[=*n*]	0, unless the file was marked for delete when it was opened.	Corresponds to the D parameter code, which specifies that the file should be deleted. /DELETE or /DELETE=n for nonzero values of n enable the parameter code and /DELETE=n for a zero value of n disables the parameter code.
/RENAME=*name* *or* /REN=*name*	Do not rename the file.	Corresponds to the R parameter code and the file name positional parameter. The R parameter code specifies that the file should be renamed and the file name positional parameter gives the new name of the file.

8

Spool Device

Caché lets you send print output directly to your printer or screen, or retain it in a spool global for printing at a later time. Caché spooling is independent of the spooling performed by your operating system.

Spooling in Caché is a technique that lets you automatically save the output of a program in the ^SPOOL subscripted global instead of printing it immediately. You can print the output later by sending the contents of the ^SPOOL global to the printer. This chapter describes two ways of using this spooling facility: using ObjectScript commands (**OPEN**, **USE**, **WRITE**, **CLOSE**), or using the **%IS** and **%SPOOL** utilities.

8.1 Opening and Using the Spool Device

To send output to the spool global in your current namespace, you open the spooler and specify it as your output device.

The spooler is a predefined device provided with Caché. It is assigned device number 2 in the device table. This device number can be used to identify the spooler device in **OPEN**, **USE**, and **CLOSE** commands.

You can access spooler device information through the Management Portal. Select **[System] > [Configuration] > [Device Settings] > [Devices]**. Here you will find both device 2 and a device named SPOOL. By default, these are both mapped to the same physical device (device 2) and have the same option values.

When you set the Caché spooler as the current device, Caché stores any output sent to Device 2 in the global ^SPOOL in your current namespace. Each line in ^SPOOL is in a separate global node.

There are two ways to open the Caché spooler and set it as the current output device:

- Issue **OPEN** and **USE** commands
- Invoke the %IS utility

8.1.1 OPEN and USE Commands for Spooling Device

You can open the spooling device directly by issuing **OPEN** and **USE** commands to that device.

```
OPEN 2:(doc_num:index)
USE 2
```

Table 8–1: OPEN Positional Parameters for Spooling

Parameter	Definition
doc_num	The number of the spool document (file) you want to open. Spool documents are stored in the ^SPOOL global. The default is 1.
index	Line number, 1 or greater, within the spool document. The default is 1.

These are positional parameters. If you omit both parameters, they default to (1:1). You can set first parameter (*doc_num*) and omit the second (*index*), which defaults to 1. However, if you set the second parameter, you should specify the first parameter.

Caché uses these values to locate the lines you want to print. It treats the *doc_num* parameter as the first subscript of the ^SPOOL global. It treats the *index* parameter as the second subscript of the ^SPOOL global.

8.1.1.1 USE Command

When you issue **USE 2** for device 2 after the command `OPEN 2:(doc_num:index)`, Caché sends any subsequent output to the spooler at `^SPOOL(doc_num:index)`. Each output line is stored as a separate global node within ^SPOOL.

8.1.1.2 WRITE Command

To write a line to the ^SPOOL global, issue a **WRITE** command, ending with a line terminator character. For example:

ObjectScript

```
/* Writing to the ^SPOOL global */
OPEN 2
USE 2
  WRITE "First line of text",!
  WRITE "Second line of text",!
CLOSE 2

/* Displaying the ^SPOOL global */
WRITE ^SPOOL(1,1),^SPOOL(1,2)
```

Each line ends with a line terminator (the exclamation mark) and is stored in a separate global node.

However, in producing a single print line, you may want to use several **WRITE** commands; if a **WRITE** does not contain a line terminator character, the next **WRITE** command appends to the same print line. Both write to the same global node. This line is held in a buffer and not written into the spool global until either a line termination character is issued, or the spooler device is closed.

The following example writes one global node when **CLOSE** is issued:

ObjectScript

```
/* Writing to the ^SPOOL global */
OPEN 2
USE 2
  WRITE "First half of line "
  WRITE "Second half of line"
CLOSE 2

/* Displaying the ^SPOOL global */
WRITE ^SPOOL(1,1)
```

The line terminator character is commonly the ! (exclamation mark) **WRITE** command code character. This is equivalent to a carriage return (ASCII 13) *and* a line feed (ASCII 10). To terminate a line, both of these control characters are necessary. Issuing just a carriage return (ASCII 13) causes the carriage return to be concatenated into the line node, rather than initiating a new line node. In Terminal, a line of this type displays as an overwrite of the text before the carriage return, by the text following it.

The following example writes only two line nodes in the ^SPOOL file:

ObjectScript

```
/* Writing to the ^SPOOL global */
OPEN 2
USE 2
  WRITE "AAAAAAAAAA",$CHAR(10),$CHAR(13)
  WRITE "BBBBBBBBBB",$CHAR(13)
  WRITE "XXXX",!
CLOSE 2

/* Displaying the ^SPOOL global */
WRITE ^SPOOL(1,1),^SPOOL(1,2)
```

For more information, see the OPEN, USE, WRITE, and CLOSE commands in the *ObjectScript Language Reference*.

8.2 Spooling and Special Variables

When writing to ^SPOOL, Caché continually updates the **$X** and **$Y** special variables. **$X** indicates the number of characters written to the current index line, and **$Y** contains the number of lines written during the current **OPEN**. Note that the value of **$Y** is not necessarily the same as the node index. For example:

ObjectScript

```
ZNSPACE "SAMPLES"
/* Writing to the ^SPOOL global */
OPEN 2:(2:3)
USE 2
  WRITE "Hello " SET x1=$X,y1=$Y,z1=$ZA
  WRITE "world",! SET x2=$X,y2=$Y,z2=$ZA
  WRITE "Good to see you",! SET x3=$X,y3=$Y,z3=$ZA
CLOSE 2

/* Displaying the ^SPOOL global */
WRITE ^SPOOL(2,3),^SPOOL(2,4)
WRITE !,"$X=",x1," ",x2," ",x3
WRITE !,"$Y=",y1," ",y2," ",y3
WRITE !,"$ZA=",z1," ",z2," ",z3
```

In this example, the first **WRITE** sets $X=6 (the current column number) and the second and third **WRITE** both set $X=0 (because of the line returns). The first **WRITE** sets $Y=0, the second $Y=1 (because of the line return), and the third $Y=2. Note however, that the lines that are being written are ^SPOOL(2,3), and ^SPOOL(2,4). To determine the index number, use **$ZA**.

Writing to a spool file sets the **$ZA** special variable with the next available index number. Thus, if you are writing to index=3, and do not include a line terminator, **$ZA**=3 (because the next WRITE continues writing to index 3), but if you do include a line terminator, **$ZA**=4.

The **USE** command sets **$ZB** to contains the *doc_num* of the spool file specified in the **OPEN** command.

Note: The **$IO** special variable is not modified by writing to a spool file. Normally, **$IO** is reset by a **USE** command to contain the ID of the current device. However, when the device is an output-only device (such as the spooler), **$IO** continues to contain the ID of the current *input* device.

For more information, see the $X, $Y, $ZA, $ZB, and $IO special variables in the *ObjectScript Language Reference*.

8.3 Closing the Spool Device

When you issue **CLOSE** for device 2, the system automatically sets the node `^SPOOL(doc_num,2147483647)` to store information about closing the spool document and the highest index number the output reaches.

8.3.1 Changing Namespaces

When you change namespaces with a SPOOL device left open, the spool device is closed automatically before the namespace change takes effect. The closing record in the *^SPOOL* global is written into the correct database.

8.3.2 Abort Job Processing

If you open a spool device, dismount the current directory, then issue a HALT command or the **Terminate($JOB)** method of the SYS.Process class, Caché returns a persistent <PROTECT> error for subsequent attempts to access this spool device. To avoid this, change the namespace to automatically closes any open SPOOL device.

8.4 Viewing the ^SPOOL Global

Like any subscripted global, you can display lines from the spool file by issuing a **WRITE** command, as follows:

ObjectScript

```
ZNSPACE "SAMPLES"
WRITE "1st spool file node: ",^SPOOL(1,1),!
```

However, to view and edit the spool file itself, go to the Management Portal and select the **Globals** option. Select your current namespace (SAMPLES in the above example), locate the SPOOL global, then click **data**. This displays spool file data similar to the following examples.

In the following spool file, the (!) termination character ends each node line in the spool file. These termination characters are part of the spool file, concatenated to the text string as a $CHAR(13,10) (Return and Line Feed).

```
^SPOOL(1,1)=<<"First line of text"_$C(13,10)>>
^SPOOL(1,2)=<<"Second line of text"_$C(13,10)>>
^SPOOL(1,2147483647)={59605,43605{3{
```

In the following spool file, there are no line termination characters. The two **WRITE** commands wrote a single node line, which was terminated by the closing the spool file.

```
^SPOOL(1,1)=First half of line Second half of line
^SPOOL(1,2147483647)={59605,43725{2{
```

In the following spool file, return and line feed characters were explicitly coded in the **WRITE** commands. The $CHAR(10) line feed character initiates a new node line, and the $CHAR(13) return character is concatenated into these node lines.

```
^SPOOL(1,1)=<<"AAAAAAAAAA"_$C(10)>>
^SPOOL(1,2)=<<$C(13)_"BBBBBBBBBB"_$C(13)_"XXXX"_$C(13,10)>>
^SPOOL(1,2147483647)={59605,44993{3{
```

The final line of the spool file is generated by Caché when you close the spool file. It consists of the literal 1,2147483647; the date and time in **$HOROLOG** format (59605,44993), and the number of lines in the spool file, including the final line.

You can edit or delete these spool file text lines. using the **data** display for the SPOOL global in the Management Portal **Globals** option.

8.5 Opening the Spooler Using the %IS Utility

%IS provides a convenient user interface at which a user can select the spool device, as well as any other device defined in the ^%IS global in the %SYS namespace. Using **%IS**, you can create a named spool file and write lines of text to that file. You can then print this spool file using the %SPOOL utility.

Note: Only spool files opened using the **%IS** utility can be manipulated using the **%SPOOL** utility.

To create a spool file using **%IS** do the following steps:

1. Invoke the **%IS** utility to open the spooler:

   ```
   >DO ^%IS
   ```

2. At the "Device" prompt enter "2" or the mnemonic "SPOOL".
3. At the "Name" prompt, enter the name of the spool document (file). (Press **Return** at the "Name" prompt if you decide not to open the spool device.) If you enter the name of an existing spool document, **%IS** asks if it is correct, displays the last line of the file, and lets you choose where to add the new information. If you enter a new name, **%IS** asks if you want to create a new document. Press **Return** to create a new spool document, or enter "No" to redisplay the "Name" prompt.
4. At the "Description" prompt, enter a one-line description. To increase readability, the description of the spooled document is on a separate line and wraps at column 70 if it is too long to fit on one line.

The following example writes the line "TESTING SPOOL FUNCTIONALITY" to the ^*SPOOL* global. *IO* is a variable that **%IS** sets to the device you specify at the "Device" prompt.

```
%SYS>DO ^%IS
Device: 2
Name: SPOOLFILE not found
Create new document 'SPOOLFILE'? Yes => <RETURN>
Description: This is my test spool file
%SYS>USE IO WRITE "TESTING SPOOLING FUNCTIONALITY",!
%SYS>CLOSE IO
```

8.6 Managing Spooled Documents Using %SPOOL

You manage spool files created when you access the Caché spool device with the **%SPOOL** utility. Caché spooling is independent from system spooling.

Spooling in Caché is a technique that lets you automatically save the output of a program in the global ^SPOOL instead of printing it immediately. You can print the output later by sending the contents of the global to the printer.

Use the **%SPOOL** utility to print, list, or delete spool documents in the ^SPOOL global in your current namespace. If you send a document to the spooler from a particular namespace, you must run the **%SPOOL** utility from that namespace to access it.

Note: Only spool files opened using the **%IS** utility can be manipulated using the **%SPOOL** utility.

%SPOOL asks which spooling option you want. You can choose any of the three functions by entering either:

- The menu number of the function
- The first letter of the function name

You can also enter a question mark (?) to display a list of these functions.

The following example shows how you select a spooling function, in this case, Print.

```
%SYS>DO ^%SPOOL

Spool function: ?

The available spool functions are:

  1) Print
  2) List documents
  3) Delete document

Enter the number or first few characters of the name of the
spool function you want.

Spool function: 1 Print
```

The following sections describe how to use the **%SPOOL** utility to perform the following tasks:

- Print spool documents
- List spool documents
- Delete spool documents

8.6.1 Printing with %SPOOL

Option 1 of the **%SPOOL** utility menu, Print, lets you print one or more documents in the ^SPOOL global on any device, resume printing an interrupted document, and handfeed single sheets of paper into a letter-quality printer. By sending output to the spooler, you release your terminal for other uses while the output device prints your document.

You can start printing either before or after the spool document is fully created. If the printer catches up to the new output, the print process pauses for five seconds, then prints all the output accumulated during that time. The print process knows when you have closed the spool document and finishes when the document is done.

As **%SPOOL** prints the document, it keeps track of the pages it has printed. It also creates a page index, so that you can sort through the document by page number and begin printing at the top of any page you choose.

If you stop printing (for example, by pressing **ctrl-c** during terminal output, or if your printer breaks), you can later resume at the top of the last partially printed page or at the top of any other page in the document. Note that Caché does not count form feeds at the start of the document as pages in the page count.

%SPOOL uses the term *despool* to mean print. There will be values in the Despool start-end column and on the description line only if the document has been printed (despooled).

8.6.1.1 Using the Print Function

1. At the "Spool function:" prompt, enter 1.
2. At the "Name:" prompt, enter a ? to display help text, enter ?? to list all existing spool documents in the current namespace, or enter the name of the spool document you want to print. **%SPOOL** confirms that this is the correct document.
3. When **%SPOOL** asks you for the page where you want to start printing, press **return** to start at the first page, or enter any page number in the document. If you try to start printing at the top of a page the printing process has not yet reached, the following message displays: **WARNING: Printing hasn't reached this point**. After this warning, **%SPOOL** asks if you are sure you want to start printing on the page you selected. If you enter No, it returns you to the " Start at page:" prompt. If you enter Yes to confirm the starting page, **%SPOOL** displays the first few lines of the page in question and reconfirms that this is the right page.
4. You are prompted for the number of copies.
5. **%SPOOL** lets you enter the names of other spool documents you want to print. When you respond to the " Name:" prompt by pressing **return**, it asks you for the output device and its right margin. Enter this information to start printing.

   Note that **%SPOOL** issues a form feed after each page, whether you are printing on a screen or a printer.

The following example shows you how to print a document in the ^SPOOL global, in this case called SPOOLFILE. The document will print on the device called MYPRINTER.

```
%SYS>DO ^%SPOOL

Spool function: 1  Print
Name: ??

#  Name        Lines   Spool start      Despool start-end
1  SPOOLFILE   1      30 Aug  2:23 pm    30 Aug  2:25 pm-2:25 pm
  This is my test spool file

Name: SPOOLFILE

1  SPOOLFILE    30 Aug 2003  2:23 pm  this is my test spool file
SPOOLFILE has 1 pages.
Is this correct?  Yes=>Y
Start at page:  1=>Y
How many copies?  1=>Y

Name:RETURN
Print spooled files on
Device: MYPRINTER RETURN Parameters: "WNS"=>
Free this terminal? Yes =>Y
Starting Job in background . . . started.

Spool function:
```

8.6.2 Listing Spooled Documents

Option 2 of the **%SPOOL** utility menu, List documents, displays a list of the documents currently spooled for the directory in which **%SPOOL** is running. If there is no Despool start-end value, the document has not yet been despooled (printed).

The description of each spooled document appears on one or more separate lines following the rest of the information about that document.

In the following example, the user selected Option 2. The display reveals two documents stored in the spooler. The first was stored on August 30 at 2:23 p.m. and printed the same day at 2:25 p.m. The second was stored on March 4 at 11:39 a.m. and printed the same day at 11:42 a.m.

```
Spool function: 2   List documents

# Name      Lines   Spool start     Despool start-end
1 SPOOLFILE  1     30 Aug  2:23 pm  30 Aug  2:25 pm- 2:25 pm
  This is my test spool file

3 LONGFILE   1     04 Mar 11:39 am  04 Mar 11:42 am- 11:42 am
  This is a very long description line that shows you what happens when you
have a long description. It shows you how the text wraps from line to line.
This particular description was made intentionally long, so as to wrap at least
twice.
```

8.6.3 Deleting Spooled Documents

Option 3 of the **%SPOOL** utility menu, Delete document, lets you delete one or more spool documents, When **%SPOOL** prompts you for a name, enter the name of the document you want to delete, or enter ?? to display the current spool documents for the namespace you are in. Enter a ? for help text.

%SPOOL confirms that this is the correct document, and that you want to delete it. If you answer "Yes," **%SPOOL** deletes the document, and allows you to name other documents you want to delete.

The following example deletes the spooled document called SPOOLFILE.

```
Spool function: 3   Delete document
Name: ??

# Name        Lines  Spool start        Despool start-end
1 SPOOLFILE     1    30 Aug  2:23 pm  30 Aug  2:25 pm- 2:25 pm
  This is my test spool file

Name: SPOOLFILE

1 SPOOLFILE    30 Aug 2003  2:23 pm  this is my test spool file
SPOOLFILE has 1 pages.
Is this correct?  Yes=>Y
Delete SPOOLFILE? No=> Y [Deleted]

Name:
```

9

Printers

This chapter discusses how to configure and use print devices in Caché. A printer is a physical output-only device. A printer may be a character printer, or a non-character device such as a fax or plotter.

In most cases, output is not sent directly to a printer. Often, output to be printed is first sent to a logical spool device (the ^SPOOL global). The contents of the ^SPOOL global can then be sent to the physical printer. For further details on spooling, refer to the Spool Device chapter of this manual.

9.1 Overview of Printers

Note that Windows and UNIX® handle printer I/O differently.

- Windows systems handle a printer as a sequential I/O device, and thus follows the same syntax as sequential file I/O. However, a printer connected through a serial communications port is handled as a terminal I/O device.
- UNIX® systems always handle a printer as a terminal I/O device. UNIX® treats it as a “character special” file on a tty device, and thus follows the same syntax as terminal I/O.

On a Windows system, you can return a count of the current printers on your system using the **%Library.Device.InstalledPrinters()** method. You can return a list of the current printers on your system using the **%Library.Device.GetPrinters()** method.

9.2 Specifying a Printer

A printer can be assigned a device number between 256 and 2047, inclusive. This range of device numbers are also used for terminals and flat files.

On a Windows system, you can refer to a printer using its device number or an assigned device mnemonic. The Windows default printer mnemonic is `|PRN|`.

There are two ways to specify a printer:

- Call the **%IS** utility, which allows you to specify the device by using a mnemonic defined in the *%IS* global. This utility opens the device and sets its parameters.
- Issue the I/O commands **OPEN**, **USE**, and **CLOSE**, using the operating system device name, specified as a quoted string.

9.2.1 Opening a Printer

When opening a printer, you can use the device name to specify the device. The device name must be enclosed in quotes. The maximum length of this device name is 256 characters. The form is as follows:

```
OPEN "device"
USE "device"
CLOSE "device"
```

On Windows, you can also have a printer attached to a serial communications port. In this case, the printer is treated the same as terminal I/O with the following syntax:

```
OPEN "comn:"
```

Where *n* is the port number to which the printer is attached.

9.2.2 Specifying a Printer on Windows

To use the default printer on Windows, enter the following:

ObjectScript

```
OPEN "|PRN|"
```

This causes Caché to use the default Windows printer for your machine, if you have set the default printer for your machine. (For information on how to set the default printer, see your Windows documentation.)

To use a printer other than the default printer, enter the following:

ObjectScript

```
OPEN "|PRN|printer"
```

printer	The Universal Naming Convention (UNC) name or a name that shows up on your machine's list of printers (Print Manager in Windows NT, 2000, and XP).

The following example illustrates the use of a UNC name:

ObjectScript

```
OPEN "|PRN|\\business\accounting"
```

The following example illustrates the use of a non-UNC name that might appear on your machine's list of printers:

ObjectScript

```
OPEN "|PRN|HP LaserJet 5P"
```

Note: InterSystems discourages the use of printer port names like COM1, LPT1, etc. If you do use such a name, Caché will try to figure out which printer, if any, that name refers to and then it will use that name. However, this will be a slow operation and is not really appropriate for Windows

On Windows systems, a printer is identified by the name on the **OPEN** command and is handled by the sequential I/O module, not the terminal I/O module. Thus the **OPEN** and **USE** command arguments supported are the same as those for sequential file I/O, *not* terminal I/O. The exception to this is a printer connected to a Windows system through a serial communications port; in this case, the printer is handled as a terminal I/O device.

On Windows systems, you cannot use the ":n" positional parameter to control the print margin used. Such notation is ignored by Caché. Code such as `"|PRN|":121` is ignored. To control the printer width, send the appropriate control characters for that printer. The notation does work on other platforms.

On Windows, **OPEN** supports most of the sequential I/O keyword parameters, as described in the Sequential File I/O chapter of this manual. The following table describes additional printer keyword parameters for controlling a printer (handled as a sequential file) on Windows systems with the **OPEN** command.

Table 9–1: Additional OPEN Keyword Parameters for Windows Printers

Keyword	Default	Description
/DOCNAME= "*name*"	"Cache"	/DOCNAME enables you to redefine the printer job name.
/OUTPUTFILE= "*filename*"	NULL	/OUTPUTFILE redirects print to a file. Specify a fully-qualified pathname.
/DATATYPE= "*type*"	"RAW"	/DATATYPE enables you to redefine the datatype of the printer spool data. One frequently-used datatype is TEXT.

On Windows systems, if the **OPEN** prints directly to the printer (does not use a logical spool device), the **OPEN** command *timeout* argument does not expire if the printer is turned off or does not exist. The Caché process remains in a suspended state until the printer becomes available, or until the print document is cancelled from the Windows Control Panel.

9.2.3 Specifying a Printer on UNIX®

To open an I/O device on a terminal that has the UNIX® device name /dev/tty06, enter the following command

ObjectScript

```
OPEN "/dev/tty06"
```

On UNIX® systems, a printer is identified by the name on the **OPEN** command and is handled as a "character special" file on a tty device. Thus the **OPEN** and **USE** command arguments supported are the same as those for terminal I/O, *not* sequential file I/O.

On UNIX®, **OPEN** supports most of the terminal I/O keyword parameters, as described in the Terminal I/O chapter of this manual.

9.3 Directing Output to a Printer

You can use the %IS utility to direct output to a printer. You can invoke the %IS utility with the command `DO ^%IS`. (You can also use `DO OUT^%IS` to specify that you are selecting an output-only device.) In either case, Caché returns the `Device:` prompt. To specify a printer, reply with either the default mnemonic "|PRN|", or the mnemonic of another configured printer. The %IS utility then suggests OPEN parameters; for a printer, the default is "W" (write-only). You can accept the parameter default by pressing **Return**, as shown in the following example:

```
%SYS>DO ^%IS
Device: |PRN|
Parameters? "W" => <RETURN>
%SYS>
```

This opens the specified printer as an output device for the current process.

The %IS utility sets various variables. The following are the printer default values on a Windows system.

Table 9–2: Variables Set by %IS

Variable	Value	Description
IO	|PRN|	Device mnemonic of selected device.
IOF	#	Form feed character. `WRITE #` issues a form feed and changes **$Y**. `WRITE @IOF` should be used to form feed.
IOBS	$C(8)	Backspace/overprint character (ASCII 8). `WRITE $CHAR(8)` issues a backspace and changes **$X**. `WRITE *8` issues a backspace but does not change **$X**. `WRITE @IOBS` should be used to backspace.
IOM	132	Right margin; line length in characters.
IOSL	66	Page length in characters.
IOT	OTH	Device type. Here “Other”.
IOST	P-DEC	Device subtype name.
IOPAR	("W")	**OPEN** parameters. Here “W” because a printer is a write-only device.
MSYS	M/WNT	Type of system (such as UNIX® or Windows NT). M/WNT is Caché on Windows NT.
POP	0	Indicates that %IS was run (and these variables initialized). If 0, a device was specified. If 1, no device was specified (user entered STOP in response to the `Device:` prompt).

Most of these values can also be viewed and set from the Management Portal. Select **[System] > [Configuration] > [Device Settings].** View the current contents of **Devices** and **Device SubTypes**. Select **Edit** to view the settings for a specific printer.

9.3.1 %IS Printer Set-Up Variable

When you call %IS for spooling, you can pass it the IOPGM variable, which specifies the name of the routine that sets up printer forms alignment.

9.4 Printer as Alternate Device

You can specify a printer as the alternate device for all processes by defining a new device named “A” and specifying a physical device of |PRN|. Then when you use %IS, specify A at the `Device:` prompt.

You can set a printer as the alternative I/O device for a terminal process. Go to the Management Portal. Select **[System] > [Configuration] > [Device Settings] > [Devices].** Select Edit for the current terminal device and specify an Alternate Device value.

10

Magnetic Tape I/O

This chapter tells you how to program I/O for magnetic tape in Caché. It describes I/O commands and special variables as they relate to magnetic tape.

- Using the Caché Magnetic Tape Handler
- Reading and Writing ANSI and EBCDIC Labeled Tapes
- Special Variables Show I/O Conditions
- Magnetic Tape Mnemonic Space for Write /mnemonic

10.1 Using the Caché Magnetic Tape Handler

This chapter describes I/O commands you use with the Caché magnetic tape handler.

With this tape handler, you refer to your magnetic tape device by device numbers 47 through 62 as configured in the Management Portal.

To set up magnetic tape device numbers and select a tape handler, see Managing Magnetic Tape Devices.

10.1.1 OPEN Command

Selects a device and specifies its parameters.

10.1.1.1 Syntax

```
OPEN devicenum:(codesparam:reclenparam:blkszparam):timeout:"mnespace"
```

where:

Argument	Definition
devicenum	The logical device number, from 47 to 62 (inclusive).

Argument	Definition
codesparam	*Optional* — 1st positional parameter: format codes. A string of one or more letter codes for magnetic tape format options. These letter codes are listed in the table Magnetic Tape Format Codes. The string of codes must be in quotation marks. Only certain combinations of codes are permitted. This optional parameter has a default of "AUV" (ASCII characters, no labels, variable-length records in ANSI D format) on Windows systems. On UNIX® systems, the default is "ALV" (ASCII characters, standard labels, variable-length records in ANSI D format).
reclenparam	*Optional* — 2nd positional parameter: record length. If *codesparam* includes F (fixed-length records), *reclenparam* is the logical record length, from 1 to 510 bytes. If records are not fixed-length, specify 0 for *reclenparam* or omit it.
blkszparam	*Optional* — 3rd positional parameter: Block size, in bytes. An even number from 16 bytes to 32KB, and a multiple of *reclenparam*. The default is 2048 bytes. The maximum value is highly platform, drive, and character-coding dependent. The recommended maximum is 32KB *characters*; larger block sizes up to 65,535 bytes are possible in some circumstances.
timeout	*Optional* — The number of seconds Caché waits for an **OPEN** to finish; a positive integer. If the timeout is zero, **OPEN** returns control to the process immediately. A timed **OPEN** sets $TEST. If the **OPEN** succeeds within the timeout period, **$TEST** is set to True (1). If the **OPEN** times out, **$TEST** is set to False (0).
mnespace	*Optional* — Name of the file that contains mnemonics and their meanings for use with the `WRITE /mnemonic` command. See the section Magnetic Tape Mnemonic Space for WRITE /mnemonic for more details.

Only the *devicenum* argument is required. You must specify a magnetic tape device by its device number. Caché reserves device numbers 47 through 62 (inclusive) for magnetic tape devices. These device numbers are defined in the Management Portal. Select **[System] > [Configuration] > [Device Settings] > [Devices]** to edit device number assignments.

Arguments are separated by colons (:). If you omit an argument, you must specify the colon. You must not end either the command or its parameter list with a colon.

The optional parameter list is enclosed in parentheses, and can specify its parameters as either positional parameters (in the order shown), or as keyword parameters with the syntax /KEYWORD=value. Keyword parameters may be specified in any order; because Caché executes them in left-to-right order, interactions between parameters may dictate a preferred order in some cases. You can mix positional parameters and keyword parameters in the same parameter list. The enclosing parentheses are required if you specify more than one parameter.

For more information, see OPEN in the *ObjectScript Language Reference*.

The three optional parameters (*codesparam*, *reclenparam*, and *blkszparam*) control data format. If you omit them, the following default parameters apply:

On Windows systems: indicating ASCII characters, no labels, variable-length records, and 2048-byte blocks:

```
("AUV":0:2048)
```

On UNIX® systems: indicating ASCII characters, ANSI-standard labels, variable-length records, and 2048-byte blocks:

```
("ALV":0:2048)
```

These default **OPEN** parameter values are defined in the Management Portal. Select **[System] > [Configuration] > [Device Settings] > [Devices]**, and then click "Edit" for a specific device (magnetic tape devices are numbered 47 through 62). When you click on a listed device, it displays the **Open Parameters** default value.

Caché ignores the data format parameters specified with **OPEN** if the device has already been opened, either by running the **%IS** utility or by issuing another **OPEN** command. You therefore must issue a **CLOSE** command for the device before you can reopen it and assign new parameters.

10.1.1.2 Specify Tape Format Codes in codesparam

The following table describes the magnetic tape format codes you can specify in *codesparam* of the **OPEN** command.

Table 10–1: Magnetic Tape Format Codes

Code	Format	Definition
A (default)	ASCII	Uses ASCII character set.
D	DOS-11	Uses DOS-11 labeling, the ASCII character set, and stream data format.
E	EBCDIC	Translates ASCII to EBCDIC on output, and EBCDIC to ASCII on input.
F	Fixed length	Assumes fixed-length records for input. Caché uses stream data format for output; you are responsible for padding records to the correct length.
K\name\ *or* Knum	I/O Translation Mode	When you use the K protocol for a device, I/O translation will occur for that device if translation has been enabled system-wide. You identify the previously defined table on which the translation is based by specifying the table's name. The "+" and "-" options for turning protocols on and off are not available with the K protocol. (The older form, "Knum," where *num* represents the number of the slot the table is loaded in, is being phased out but is still supported. The system manager can display slot numbers in the NLS utility in the selection window for each table type.)
L (default UNIX®)	Standard labels	Uses ANSI standard labels if you select the ASCII character set; uses IBM standard labels if you select the EBCDIC character set.
R	Read mode	Acts as a logical protection level during write operations. If you try to write to a tape that has Read mode specified or has a write-protect ring, an error occurs. Ignored for Read commands, except you must include this parameter if the tape has no write-protect ring. Write mode is the default. R is required for write-protected tapes on some UNIX® systems.
S	Stream format	On output, packs characters, including <RETURN> and, sequentially into the buffer. On input, treats <LINE FEED>,, and <FORM FEED> as string delimiters, and ignores <RETURN> and null characters. Jobbed processes that inherit TCP devices are automatically set to S format. You can reset the format with the **USE** command.
U (default Windows)	Unlabeled	Unlabeled. If you will be reading from an unlabeled tape, you must open it with U.
V	Variable length	If you select ASCII characters, uses ANSI D format; if you select EBCDIC, specifies EBCDIC V format. Each argument of a **READ** or **WRITE** corresponds to one logical record.

Code	Format	Definition
Y\name\ *or* Ynum	$X/$Y Action Mode	When you use the Y protocol for a device, the system uses the named $X/$Y Action Table. You identify the previously defined $X/$Y Action Table on which translation is based by specifying the table's name. You can get the name from the system manager. $X/$Y action is always enabled. If Y is not specified and a system default $X/$Y is not defined, a built in $X/$Y action table is used. The + and – options for turning protocols on and off are not available with the Y protocol. (The older form, Y*num* where *num* represents the number of the slot the table is loaded in, is being phased out but is still supported. The system manager can display slot numbers in the %NLS utility in the selection window for each table type.)

The following table shows the magnetic tape density codes that you can specify in *codesparam* of the **OPEN** command.

Table 10–2: Magnetic Tape Density Codes

Code	Format	Definition
1	800 BPI	Selects 800 BPI (Bits per inch) recording density for drives that support multiple densities.
2	1600 BPI	Selects 1600 BPI for drives that support multiple densities.
3	6250 BPI	Selects 6250 BPI for drives that support multiple densities.

10.1.1.3 Allowed Combinations of Format Codes

The following table shows which combinations of *codesparam* format codes you can specify. If you include conflicting codes in the parameter string, the last code prevails.

Table 10–3: Allowed Combinations of Format Codes

	A	D	E	F	L	R	S	U	V
A ASCII	-	Y	N	Y	Y	Y	Y	Y	Y
D DOS	Y	-	N	N	N	Y	Y	N	N
E EBCDIC	N	N	-	Y	Y	Y	Y	Y	Y
F Fixed Length	Y	N	Y	-	Y	Y	N	Y	N
L Labeled	Y	N	Y	Y	-	Y	Y	N	Y
R Read Mode	Y	Y	Y	Y	Y	-	Y	Y	Y
S Stream Format	Y	Y	Y	N	Y	Y	-	Y	N
U Unlabeled	Y	N	Y	Y	N	Y	Y	-	Y
V Variable Length	Y	N	Y	N	Y	Y	N	Y	-

10.1.1.4 OPEN Keyword Parameters

The following table describes the keyword parameters for controlling magnetic tape devices via the **OPEN** command.

Table 10–4: OPEN Command Keywords for Magnetic Tape Devices

Keyword	Default	Corresponding Positional Parameter	Description
/ASCII *or* /ASC	Use ASCII	*codesparam* =“A”	Specifies that the ASCII character set be used.
/BLOCKSIZE=*n* *or* /BLO=*n*	No default	*blkszparam*	Specifies the size of a tape block.
/DENSITY=*n* *or* /DEN=*n*	No default	*codesparam* =“1” *or* *codesparam* =“2” *or* *codesparam* =“3”	Tape density. The value of *n* specifies the tape density in BPI. Recognized values are 0, 800, 1600, and 6250. A value of 0 selects the default density for the device. Other values of *n* are ignored.
/DOS	No default	*codesparam* =“D”	Specifies DOS-11 labeling, stream format, and the ASCII character set.
/EBCDIC *or* /EBC	No default	*codesparam* =“E”	Specifies ASCII to EBCDIC translation on output and EBCDIC to ASCII translation on input.
/FIXED *or* /FIX	Not fixed	*codesparam* =“F”	Specifies fixed-length records for input. (The programmer is responsible for padding records to the correct length for output.)

Keyword	Default	Corresponding Positional Parameter	Description
/GZIP[=*n*]	1		Specifies GZIP-compatible stream data compression. /GZIP or /GZIP=*n* (for nonzero values of *n*) enables compression on WRITE and decompression on READ. /GZIP=0 disables compression and decompression. Before issuing /GZIP=0 to disable compression and decompression, check the $ZEOS special variable to make sure that a stream data read is not in progress. /GZIP compression has no effect on I/O translation, such as translation established using /IOTABLE. This is because compression is applied after all other translation (except encryption) and decompression is applied before all other translation (except encryption).
/IOTABLE[=*name*] *or* /IOT[=*name*]	If *name* is not specified, the default I/O translation table for the device is used.	*codesparam* =“K\name\”	Establishes an I/O translation table for the device.
/LABELED[=*n*] *or* /LAB[=*n*]	L format is the default on UNIX® platforms; U format is the default on Windows platforms.	*codesparam* =“L” *or* *codesparam* =“U”	/LABELED and /LABELED=*n* for nonzero values of *n* corresponds to the L format code, which specifies standard labels (ANSI standard labels if the ASCII character set is used, or IBM standard labels if the EBCDIC character set is used). /LABELED=n for a zero value of *n* corresponds to the U format code, which specifies an unlabeled tape.
/PARAMS=*str* *or* /PAR=*str*	No default	*codesparam*	Corresponds to the *codesparam* format code string positional parameter. (This keyword provides a way to specify a format code string in a position- independent way.)
/READ	No default	*codesparam* =“R”	Acts as logical Write protection for the tape. (If a **WRITE** operation is attempted, an error will occur.)
/RECORDSIZE=*n* *or* /REC=*n*	No default	*reclenparam*	Specifies the logical record length when fixed-length records are specified on input.

Keyword	Default	Corresponding Positional Parameter	Description
/STREAM *or* /STR	0	*codesparam* =“S”	Specifies stream record format.
/TRANSLATE[=*n*] *or* /TRA[=*n*]	1	*codesparam* =“K”	/TRANSLATE or /TRANSLATE= *n* for nonzero values of *n* enable I/O translation for the device. /TRANSLATE= *n* for a zero value of *n* disables I/O translation for the device.
/VARIABLE *or* /VAR	No default	*codesparam* =“V”	Specifies variable length record format (ANSI D format if the ASCII character set is used, or EBCDIC V format if the EBCDIC character set is used). (This is the default.)
/XYTABLE[=*name*] *or* /XYT[=*name*]	If *name* is not specified, the default $X/$Y action table for the device is used.	*codesparam* =“Y\name\”	Establishes a $X/$Y action table for the device.

10.1.1.5 Examples

The following command opens logical device 47 with the default parameters.

ObjectScript

```
OPEN 47
```

The following command opens logical device 48 with ANSI standard D format, which consists of labeled files with variable-length ASCII records.

ObjectScript

```
OPEN 48:"AVL"
```

The following command opens logical device 47 with EBCDIC unlabeled fixed-block records, 80 bytes long, in 8000byte blocks.

ObjectScript

```
OPEN 47:("EUF":80:8000)
```

10.1.1.6 Check Parameters if an OPEN Command Fails

If an **OPEN** command fails, use these hints to resolve the problem:

- Run the DEVICE utility. Make sure a device number is not being retranslated incorrectly via its relative order.

- On Windows and UNIX® systems: check the configuration settings in the Management Portal. Make sure that they show the correct physical device name or logical name for the device you have specified in the **OPEN** command.
- Check that you have mounted the correct tape drive at the operating system and Caché levels.
- Check that the block size you selected in *blkszparam* of the **OPEN** command conforms to the restrictions of your operating system. Some devices require certain block sizes that do not match the Caché defaults. Consult your operating system documentation for details.
- Check the remaining parameters. Be sure they correctly specify the device you want to open.

Because **OPEN** commands do sometimes fail, you should always specify a timeout, so that a failed **OPEN** does not cause your process to hang.

10.1.2 USE Command

Sets a previously opened magnetic tape drive as the current device.

10.1.2.1 Syntax

```
USE devicenum
```

For more information, see USE in the *ObjectScript Language Reference*.

10.1.2.2 Example

The following command makes device 47 your current device.

ObjectScript

```
USE 47
```

10.1.3 READ Command

Reads from the current device, according to the command format you specify. After the **READ**, the variable holds the contents of the record at the current position on the tape.

You control the tape head location with `WRITE *` control code options, as explained in the next section.

10.1.3.1 Syntax

```
READ variable
READ *variable
READ variable#length
```

For more information, see READ in the *ObjectScript Language Reference*.

10.1.3.2 Tape Without Write-Protect Ring

To read a tape without a write-protect ring, you must **OPEN** the tape with the "R" parameter.

Note: On *some* UNIX® systems, R is required for write-protected tapes.

10.1.4 WRITE Command

Writes data to the current location on the current magnetic tape device.

10.1.4.1 Syntax

```
WRITE *-n
WRITE /mnemonic
```

For more information, see WRITE in the *ObjectScript Language Reference*.

Table 10–5: Magnetic Tape WRITE Options

Argument	Definition
**-n*	Specifies a control code; see "WRITE *-n Controls Magnetic Tape Functions" for details.
/mnemonic	Causes Caché to interpret mnemonic as defined in the active mnemonic space. If there is no active mnemonic space, an error results.

You can specify an active mnemonic space in two ways:

- By naming a default mnemonic space for each device type in the Namespace and Network Configuration editor.
- By including the mnemonic parameter in the **OPEN** or **USE** command for the device.

See the section "Controlling Devices with Mnemonic Spaces" for more information on mnemonic spaces.

10.1.4.2 WRITE *-n Controls Magnetic Tape Functions

You can change the location of the tape head and write special marks and labels by using the **WRITE *** command followed by a negative integer control code. For example:

ObjectScript

```
WRITE *-5
```

The following table describes the control codes:

Table 10–6: WRITE *-n Control Codes

Code	Function	Effect
-1	Backspace	Backspaces tape one block. (Not supported for cartridge tape.)
-2	Forward Space	Advances tape one block (skipping labels) so that `WRITE *-2` at the beginning of the tape (BOT) skips volume and header labels as well as the first data block. If you use the U format code in your **OPEN** but the tape really is labeled, `WRITE *-2` at the beginning of the tape (BOT) skips the volume label. (Not supported for cartridge tape.)
-3	Write Tape Mark	Writes a tape mark. If the buffer contains unwritten data, that data is written out first.

Code	Function	Effect
-4	Write Block	Writes out the current buffer. If the format specifies labels and the tape is at BOT, header labels of the appropriate format are written before the data block.
-5	Rewind	Rewinds the tape. Any unwritten data (in addition to trailer labels and tape marks required by the format) is written before rewinding occurs.
-6	Read Block	Reads the next block. If the tape is at BOT, skips header labels before reading.
-7	Read Label	Reads the next block whether or not it is a label. (Not supported for cartridge tape.)
-8	Write Header Label	If the format specifies labels, writes header labels. If the tape is at BOT, writes a volume label before the header labels. Note that on UNIX® systems this is not supported for cartridge tape.
-9	Write EOF Label	Writes any unwritten data. If the format specifies labels, writes the appropriate trailer label. Writes two tape marks and backspaces over the second. (Not supported for cartridge tape.)

The following example opens tape device #47 and sets it as the current device. Then it rewinds the tape.

ObjectScript

```
OPEN 47
USE 47
WRITE *-5
```

Note: You can use the mnemonic parameter to accomplish many of the functions you carry out with `WRITE *-n`.

10.1.5 CLOSE Command

Closes the magnetic tape device, making it available to other users. It does not rewind the tape.

10.1.5.1 Syntax

```
CLOSE devicenum
```

For more information, see CLOSE in the *ObjectScript Language Reference*.

10.1.5.2 How Caché Closes and Rewinds Tapes

Caché performs these steps when it is writing a tape and the program issues:

- A **CLOSE** command
- A `WRITE *-9` (write EOF label) command
- A `WRITE *-5` (rewind) command

1. If data in the magnetic tape buffer has not yet been written to tape, Caché writes the buffer to tape. In the EBCDIC formats, a short block is written, if necessary, in accordance with the IBM tape standard.
2. If the tape is an ANSI- or EBCDIC-labeled tape and the last operation to the tape was a write, Caché writes the appropriate EOF label.

3. Caché writes two tape marks, then backspaces over the second tape mark, leaving the write head positioned between the tape marks.
4. Caché executes the **CLOSE** or rewind command.

10.2 Reading and Writing ANSI and EBCDIC Labeled Tapes

Caché allows you to read and write ANSI-standard and EBCDIC labels at the beginning and end of tape files.

Note: On Windows and UNIX®, you can write ANSI-standard and EBCDIC labels on 9-track tape only. You cannot label cartridge, 8mm, or 4mm tapes.

To write a single file to a labeled tape during normal usage, simply open and use the tape. Issue a `WRITE *-5` to rewind the tape, write the data, and close it.

With ANSI-standard or EBCDIC labels, Caché writes the following:

1. A volume label at the beginning of the tape.
2. A header label that consists of the two blocks, HDR1 and HDR2.
3. A tape mark.
4. The data blocks.
5. A trailer label that consists of the two blocks, EOF1 and EOF2. (The trailer label and EOF blocks are written as part of the **CLOSE**.
6. Two tape marks. These labels are automatically rewritten by Caché.

   Thus, a tape with a single file would look like:

```
VOL1, HDR1, HDR2, TM, Data, TM, EOF1, EOF2, TM, TM
```

Labels have the formats shown below. The format for UNIX® and Windows ANSI-Standard and EBCDIC labels is:

```
Volume label (Vol 1)                          |

    |VOL1CACHE1                              D%B|
    0   10   20      30
    |444400100120                            1|
    40  50   60      70

Header one (HDR1)
   |HDR1CACHE.SCR CACHE10001fseq0001020    1|
   0   10   20       30
   |0 00000 00000 00000 Open |
   40  50   60       70

Header two (HDR2)
   |HDR2fblk         M                       |
   0   10   20       30
   | 00
   40  50   60       70

Trailer one (EOF1)
   |EOF1CACHE.SRC CACHE100001fseq00010      |
   0   10   20        30

   |0 00000 00000 blkcnt Open                |
   40  50   60        70

Trailer two (EOF2)
```

```
|EOF2fblk.rec      M                    |
0   10   30        30
| 00                                     |
40  50   60        70
```

10.2.1 DOS Labels

If you write a tape with DOS labels, a single header label appears before each file, and each file ends with a tape mark. The header label consists of the file name "CACHE 001" in radix 50 notation, followed by three bytes with values 1, 1, 23. The rest of the 80-byte label is padded with ASCII nulls.

10.2.2 Record Structure

In ASCII variable-length data format, each record is preceded by a four- character number whose value is the record length plus four. The last record in the block is followed by the caret character (^). Each argument of a **WRITE** or **READ** corresponds to one record.

In EBCDIC variable-length data, the format is the same as ASCII variable-length except that each block begins with a four-character number indicating the block length.

10.2.3 File Structure

To write multiple files on a tape, you direct the placement of labels for the second and subsequent files. The volume label appears only at the beginning of the tape. The two header label blocks and a tape mark precede each file, and the two trailer label blocks and a single tape mark follow all but the last file. Use a double tape mark at the end of the last file.

You can use the magnetic tape control codes `WRITE *-8` (write header label), and `WRITE *-9` (Write EOF label) to control the placement of additional labels.

10.2.4 Creating Files on a Labeled Tape

The following table shows the steps you follow in creating three files on a labeled tape using the Caché tape handler.

Table 10–7: Creating a Three-File Labeled Tape

Step	Control Code	Definition
1	OPEN 47:"ALV" USE 47 WRITE *-5	Opens and rewinds the tape.
2	USE 47 WRITE ...	Writes the first file. Caché automatically precedes the first file with the volume label and header label.
3	USE 47 WRITE *-9	Writes the final data block, a tape mark, an EOF label, and two tape marks. Backspaces over the second tape mark.
4	USE 47 WRITE *-8 WRITE ...	Writes a header label for the second file and writes the second data file.
5	USE 47 WRITE *-9	Writes its final data block, its EOF label, and backspaces over the second tape mark.
6	USE 47 WRITE *-8 WRITE ...	Writes the third file's header label and data.
7	USE 47 WRITE *-5 CLOSE 47	Writes the last data block and a trailer label with two tape marks, rewinds the tape, and closes the magnetic tape.

10.3 Special Variables Show I/O Conditions

The special variables **$ZA** and **$ZB** contain information about magnetic tape operations. If your program requires the values of **$ZA** and **$ZB**, it should examine them after each magnetic tape operation, recognizing that an error trap will occur after some operations.

10.3.1 $ZA Holds Magnetic Tape Status

Magnetic tape errors and special conditions appear in the special variable *$ZA*. Caché updates **$ZA** after each ObjectScript command that refers to the magnetic tape device.

The following table shows the meanings of the bits in **$ZA**. The letter Y in the Trap column means that this condition causes a <MAGTAPE> error. If you have set **$ZTRAP**, this condition invokes the error code that you referred to with **$ZTRAP**. See the $ZA and $ZTRAP special variables in the *ObjectScript Language Reference*.

Table 10–8: $ZA Bits

Bit	Value	Trap	Meaning
0	1	Y	Logical Error - mixed Reads and Writes—To switch between reading and writing, either close and then open the device, or issue a Forward Space, Backspace, or Rewind command.
2	4	N	Write Protected—Attempting to open a write-protected 9-track tape without the read-only parameter sets bit 2 and opens the tape as read only. No error occurs.

Bit	Value	Trap	Meaning
3	8	Y	Error Summary—The error summary is the logical OR of all conditions that cause an error (all conditions marked “Y” in the table).
5	32	N	Beginning of Tape [BOT]
6	64	N	On Line
7	128	Y	Controller or drive error
10	1024	Y	End of Tape [EOT]
12	4096	Y	Parity or CRC Error
14	16384	Y	Tape Mark—Caché sets the Tape Mark bit when it encounters a tape mark on Read, Read Block, Forward Space or Backspace. This sets the Error Summary bit and traps to **$ZTRAP** only on Read, Read Label, and Read Block.
15	32768	Y	Tape Not Ready

Some bits indicate error conditions, while other bits indicate conditions that do not necessarily produce an error. If your program must be aware of these non-error conditions, it must test the appropriate bit of **$ZA** after every magnetic tape operation. For example, a program that might write off the end of the tape must check bit 10, End of Tape.

To test a bit, divide **$ZA** by the value for that bit listed in the above table and perform a modulo 2 operation. For example, to check if bit 14, Tape Mark, is set, enter:

ObjectScript

```
USE 47
IF $ZA\16384#2 {
   DO Endfile }
ELSE { QUIT }
```

where 16384 is 2 to the 14th power and #2 is the modulo 2 operation. Since any number to the 0 power equals 1, you do not need a divisor to check bit 0, Logical Error. Simply enter:

ObjectScript

```
USE 47
GOTO LogErr:$ZA#2
```

10.3.2 $ZB Holds Information about Driver Buffer

$ZB contains the number of bytes of data remaining in the magnetic tape driver's internal buffer. Immediately after you read a block, this number is the block size. As **READ** commands transfer logical records from the buffer to ObjectScript variables, the number in **$ZB** decreases until the next block enters the buffer.

When you write to tape, **$ZB** shows the number of unused bytes remaining in the driver's internal buffer. Immediately after you write a block, this number is the buffer size that the **OPEN** command specified. As **WRITE** commands transfer logical records from ObjectScript variables into the buffer, the number in **$ZB** decreases until Caché writes the buffer's contents in the next block.

Most magnetic tape programs need not be concerned with **$ZB**; **$ZB** is useful for programs that must deal with unusual formats and variable-length blocks. See $ZB in the *ObjectScript Language Reference*.

10.3.2.1 Example

This example checks **$ZA** and **$ZB** after each magnetic tape read, and sets *MTERR* when either of these variables indicates an error. It also sets **$ZTRAP** when a magnetic tape error occurs.

ObjectScript

```
  ; $$MTIN(MTDEV) = the next logical record read from
  ; magtape device MTDEV.
  ; Also returns ZA=$ZA and ZB=$ZB
  ; On a magtape error, MTERR=1 and $$MTIN(MTDEV)=""
  ; Expects the caller to have set $ZTRAP to trap other
  ; errors.
  ;
MTIN(IO)
     NEW REC,CURDEV
     SET MTERR=0,CURDEV=$IO,$ZTRAP="MTIERR"
     USE IO
     READ REC
MTIEXIT
     SET ZA=$ZA,ZB=$ZB
     USE CURDEV
     QUIT REC
  ;
MTIERR
      IF $ZE'["<MAGTAPE>" {
          USE CURDEV
          ZQUIT 1
          GOTO @$ZT }
  ; Use caller's error trap.
     SET $ZT="",MTERR=1,REC=""
     GOTO MTIEXIT
```

10.4 Magnetic Tape Mnemonic Space for WRITE /mnemonic

Caché provides a routine named **^%XMAG** which contains the default mnemonic space for magnetic tape devices. Using this default allows you to use the mnemonics defined in the following table to issue commands to your tape drive using the form:

ObjectScript

```
   WRITE /mnemonic
```

Table 10–9: ^%XMAG Magnetic Tape Mnemonic Space

Mnemonic	Description
INIT(%volume,%density, %format)	Write ANSI-standard tape header label with specified density (if software-settable) and format. The volume parameter is not used. Permissible density values are: 800, 1600 (default) and 6250. See Magnetic Tape Format Codes for format values. “ SA” (stream, ANSI) is the default.
MOUNT(%device,%volume)	Not implemented.
DISMOUNT(%unload)	Not implemented.
SEEKFILE(filename)	Not implemented.
REWIND	Rewind tape volume to beginning.

Mnemonic	Description
ENDFILE	Write end-of-file mark.
NEWFILE	Create new file by writing header label. If tape at BOT, writes a volume label before the header label.
TAPEMARK	Write tapemark. If the buffer contains unwritten data, that data is written out first.
SKIPBLOC(%1)	Advance the tape %1 block(s). %1 must be a positive integer.
SKIPMARK(%1)	Advance or back up tape %1 tapemark(s). If %1 is greater than zero, go forward; if %1 is less than zero, go backward.
SKIPFILE	Not implemented.
BLOCKSIZ(%bsize)	Close magnetic tape device and open it with block size set to *bsize*.
FIXED(%rsize)	Close magnetic tape device and open it with the “F” parameter and record length set to *rsize*.