
# CHARWIN: Character-based Windowing API

Version 2018.1
2024-11-07

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

*CHARWIN: Character-based Windowing API*
PDF generated on 2024-11-07
InterSystems Caché® Version 2018.1


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

# 1

# About This Book

See the Table of Contents for a detailed listing of the subjects covered in this document.

This book is a guide to CHARWIN, the Caché character-based windowing API. The following topics are covered:

- Introduction
- Commands and Functions
- Terminal Functions and Control Characters
- The Terminal Capabilities Database
- Creating and Editing Terminal Descriptions
- Creating and Editing Border Maps

For general information, see *Using InterSystems Documentation*.

# 2

# Introduction

The application programming interface (API) described here allows programmers to build windowing interfaces on character-based terminals using a mnemonic space syntax.

This interface provides:

- *Terminal independence* — Users around the world use different terminal types, so it is not efficient to use the control sequences of a particular model. Even ANSI-like terminals such as DEC's VT line are not universal. This API relies on a terminal capability database similar to that provided by termcap/terminfo in UNIX®.
- *Compatibility with existing NLS features* — Character set, $X/$Y action table, Input/Output translation tables.
- *Compatibility with existing output buffering scheme* — Terminal output is accumulated in a buffer and only sent to the line at some predefined situations like the READ and HANG commands.
- *Compatibility with existing windowing APIs* — CharWin is based on mnemonic space syntax and is a superset of both DTM and Ipsum APIs. The syntax used by Extensao depends on a Z command (ZWINDOW) and is not directly source compatible with the current implementation. However, it is easily translated to the new syntax, since all the Extensao functionality is present in this implementation.
- *Feature activation by command only* — By default, the character windows feature is disabled and the system has no overhead either in terms of memory usage or CPU cycles. Only when the programmer issues the appropriate command to enable this feature for the current terminal does the system allocate resources and traps I/O terminal operations. The feature may also be disabled at any time.

## 2.1 Definitions

The following concepts are important for a clear understanding of the character windows API:

**window**

A rectangular area on the screen. It may have an optional border (with an optional caption) around it. A window must be opened and closed by command. When it is opened, the client area is cleared, the border is optionally drawn and the cursor is positioned at the upper left corner. READ and WRITE commands are restricted to the client area. When the window is closed, the screen characters that occupied its area before it was opened are restored.

### frame

The actual region inside a window where READs and WRITEs are confined. When the window is opened, the frame occupies the whole client area. By command (/WLIM), the user may restrict the frame to a smaller rectangle inside the window. It may then grow larger again, but not larger than the original client area.

### client area

The rectangle inside the window border. If the window has no border, the client area is the whole window.

### attribute

An attribute is one of the status flags set for a character on the screen, controlling HIGHLIGHT, UNDERLINE, REVERSE, and BLINK. If the terminal is of an older type and does not keep a separate attribute byte for each character (if it stores attributes as invisible characters, for example) it cannot be used with this implementation.

### terminal capabilities

A set of terminal characteristics, including the number of lines and columns and various control sequences. Control sequences perform functions like cursor positioning and erasing to the end of the line. The cursor position and the clear screen sequences are indispensable to this API.

### z-order

The sequence in which windows are stacked on the screen. Initially this corresponds to the order in which the windows were opened. This can be changed by the /WUSE command. A window may be above another one in z-order (meaning it is placed at a higher coordinate of a hypothetical z-axis coming out of the screen) but not overlap it visually.

## 2.2 Syntax

The syntax makes it easy to port programs written in DTM, Ipsum MUMPS, and Extensao SuperMUMPS. It is based on the mnemonic space syntax, so the first command to be issued is USE. For example, the following command uses terminal device 0 and activates the %CHARWIN routine as the current mnemonic space:

```
USE 0::"^%CHARWIN"
```

This and future WRITEs / are translated to DOs to corresponding labels in this routine.

Users migrating from DTM may also consider using instead the mnemonic namespace %XDTM, which provides more compatibility with the DTM commands in MUMPS.DVF. For example, the following command uses terminal device 0 and activates %XDTM2 as the current mnemonic space:

```
USE 0::"^%XDTM2"
```

## 2.3 Input/Output Translation

The character windows API does not perform any input translation. If any keyboard mapping is desired, you must provide it using the NLS tables.

The same is true for the output side. No built-in translation is performed on the text written in windows, except for border characters. This feature is included as a convenience, because most users do not need to deal with the complexities of NLS

to create nice looking windows. It can be disabled by setting the "Disable Border Mapping" boolean flag attribute. Then the user should provide the mapping from the internal border character codes (see Border Characters and Codes) to the desired external representation using NLS.

# 3

# Commands and Functions

## 3.1 Commands

All of these commands use the WRITE /<mnemonic> syntax. For basic information on this syntax, see the Overview of I/O Commands in the *Caché I/O Device Guide*, and the WRITE command in the *Caché ObjectScript Reference*.

### /INIT(type)

This command tells the system which kind of terminal will be used. It must be one of the terminals listed in the ^%SYS("tercap") global (Caché comes with some of the more common types already defined but the user may create descriptions for other terminals as well). Issuing this command will trigger the following actions:

- Load a local table with the terminal capabilities
- The initialization string is sent to the terminal so that it behaves in a predictable way, compatible with the character windows mode.

Here are some examples:

```
W /INIT ; same as W /INIT("Terminal")
W /INIT("vt100")
W /INIT("vt320")
W /INIT("SCO ANSI")
```

When you have finished working with characters windows and want to restore the terminal to its state prior to the /INIT, the /END command should be issued .

### /WMODE(mode)

After establishing the mnemonic space and specifying a terminal type, one must set the proper compatibility mode for the character windows subsystem, according to the table below.

| Mode | Meaning |
|---|---|
| -1 | Disable the character windows subsystem and continue operating without this feature. This is a termination mode. |
| 0 | DataTree/DTM compatibility mode. |
| 1 | Ipsum MUMPS compatibility mode. |
| 2 | Extensao SuperMUMPS compatibility mode. |

If you change the mode with windows open:

- All windows are closed.
- If the new mode is not -1, the base window is opened. It fills the whole screen and has no border.

### /WOPEN

*Format:*

```
/WOPEN([wid,]col,line,width,height,border,
   attr,battr,caption,cpos,wfc,wbc,bfc,bbc)
```

Open a window with the following parameters:

| Parameter | Values |
|---|---|
| wid | Window identification (> 0) |
| | *Ipsum mode:* The programmer must provide a unique numeric ID > 0 for each window. |
| | *Other modes:* The system chooses an internal window ID and so the parameter is not present. |
| col | Character column on the screen for the left edge of the client area. |
| line | Line on the screen for the top of the client area. |
| width | Client area dimensions: width in character columns. |
| height | Client area dimensions: height is number of lines. |
| border | Border style, according to DTM conventions: |
| | 0 - none |
| | 1 - single line |
| | 2 - double line |
| | 3 - bold line |
| | 4 - blocks |
| | 5 - light dots |
| | 6 - dark dots |
| | 7 - light bar |
| | 8 - medium bar |
| | 9 - heavy bar |
| | 10 - ASCII graphics ('-', '\|' and '+') |
| | Default = 0. The actual characters that will show up in the terminal depend on the output translation table and on the codes chosen for the border characters by the user (more later). |
| attr | Window attribute. |
| | Possible attributes are "/hon", "/ron", "/uon", "/bon", "/hoff", "/boff", "/uoff", and "/roff". See Terminal Functions for details. Default = "" (none). |

| Parameter | Values |
|---|---|
| battr | Border attribute. Same conventions as attr. |
| caption | Window caption (does not appear if border style is none (0)). Default = "" (none). |
| cpos | Caption position: |
| | "tl" Top left |
| | "tc" Top center |
| | "tr" Top right |
| | "bl" Bottom left |
| | "bc" Bottom center |
| | "br" Bottom right |
| | "l" Bottom and top left |
| | "c" Bottom and top center |
| | "r" Bottom and top right |
| | Default = "". If both a top and a bottom caption are desired, then the caption parameter is of the form "top caption:bottom caption". |
| wfc | Window foreground color. Default = current color. |
| wbc | Window background color. Default = current color. |
| bfc | Border foreground color. Default = current color. |
| bbc | Border background color. For all color parameters, black is assumed to be 0 and white 7. Default = current color. |

### /WCLOSE

Closes the active window (always the topmost). The screen characters hidden by this window are restored. The next window in z-order is made active.

### /WUSE(wid)

Selects window *wid* to be the active window. This window is brought to the surface even if it has been hidden by other windows.

### /WBOX

*Format:*

```
/WBOX(col,line,width,height,border,attr,battr,caption,cpos,wfc,wbc,bfc,bbc)
```

Draws a rectangle as if a new window was being opened. This is, however, only a visual effect, and a /WCLOSE should not be called to "close" this pseudo-window. Coordinates are relative to the current frame in the active window. Other parameters are the same as those of /WOPEN.

### /WLIM

*Format:*

```
/WLIM(col,line,width,height,border,attr,battr,caption,cpos,wfc,wbc,bfc,bbc)
```

Defines new limits (a new frame) for the active window, normally for temporary use. A new border may be drawn for the new area, if and only if the active window also has a border. The values of col and line are relative to the current window. To restore the initial limits of the active window, call /WLIM.

**/BOX**

*Format:*

```
/BOX(x1,y1,x2,y2,border,battr,caption,cpos,char,wfc,wbc,bfc,bbc)
```

Draws a box with the upper left corner at (x1,y1) and lower right corner at (x2,y2), filled with character char. The corner positions are relative to the current frame in the active window. The border style and other parameters are the same as described under /WOPEN. If the char parameter is omitted, no filling is performed.

**/FILL**

*Format:*

```
/FILL(x1,y1,x2,y2,char,wfc,wbc)
```

Fills the rectangle with upper left corner at (x1,y1) and lower right corner at (x2,y2) with character char and color wfc, wbc.

**/WREFRESH**

Forces a screen update. May be called if the programmer wants to see the individual effects of /WOPEN and /WCLOSE commands, which otherwise would be combined and only the final image displayed.

**/END**

Contains a control sequence to be sent to the terminal when exiting Caché. Use this to restore the terminal to its configuration prior to starting Caché. Also frees the terminal capabilities table and disables the I/O translation. If the previous translation table needs to be restored, it must be done by explicit programming. It must be saved before calling /INIT and restored after /END.

# 3.2 Window Status

You can use the $$ZWSTATUS^%CHARWIN() extrinsic function and $$ZWSTATUS^%CHARWIN extrinsic special variable to obtain information about windows in use.

## 3.2.1 $$ZWSTATUS^%CHARWIN(wid) Function

This function works the same in all language modes. It returns the status of the window whose identification number is *wid*. If the window specified by *wid* is not open, then all fields return -1.

The status is described by a string containing the following fields, separated by semicolons:

```
col;line;width;height;border;attr;battr;curcol;curline;curattr
```

The fields specify the following values:

| | |
|---|---|
| col;line | Upper left corner of the window. |
| width;height | Window dimensions. |
| border | Border style. |
| attr | Window attribute. |
| battr | Border attribute. |
| curcol;curline | Current cursor position. |
| curattr | Current character attributes. |

**Note:** The attribute fields return strings such as "/hon", "/aoff", and so on. See /WOPEN for a list of attributes.

## 3.2.2 $$ZWSTATUS^%CHARWIN special variable

The $$ZWSTATUS^%CHARWIN special variable value varies depending on language mode.

- *DTM mode* — returns a null string.
- *Ipsum mode* — returns the number of the current window, or zero if no window is open.
- *SuperMUMPS mode* — returns a string of fields separated by commas, equivalent to the $ZWINDOW SuperMUMPS special variable:

```
winline,wincol,winheight,winwidth,limline,limcol,limheight,limwidth,
curline,curcol,curst,border,attr,visib,proc,wid
```

The fields specify the following values:

| | |
|---|---|
| *winline,wincol* | Upper left corner of the active window. |
| *winheight,winwidth* | Active window dimensions. |
| *limline,limcol* | Upper left corner of the current limits, relative to the window area. |
| *limheight,limwidth* | Current limits dimensions. |
| *curline,curcol* | Current cursor position. |
| *curst* | Cursor status (0=off, 1=on) |
| *border* | Border style. |
| *attr* | Current video attributes. The attribute fields return strings such as "/hon", "/aoff", and so on. See /WOPEN for a list of attributes. |
| *visib* | Window visibility (0=invisible, 1=visible). |
| *proc* | Current process number. |
| *wid* | Active window identification number. |

# 4

# Terminal Functions and Control Characters

All of these functions and characters are used with the standard WRITE /<mnemonic> syntax. For basic information on this syntax, see the Overview of I/O Commands in the *Caché I/O Device Guide*, and the WRITE command in the *Caché ObjectScript Reference*.

## 4.1 Terminal Functions

The following terminal functions are used for text formatting and editing. They work inside windows and with WMODE(-1) as well. The standard WRITE options (*, #, ! and ?n) work as expected inside windows.

| Mnemonic | Action |
|---|---|
| /bell | Rings the bell |
| /boff | Blink off |
| /bon | Blink on |
| /bs(n) | Backspace n positions (stop at upper left corner) |
| /clr | Clear the window. |
| /color(f,b) | Set foreground (f) and background (b) colors (0-7) |
| /cub(n) | Cursor backward n columns (stop at left margin). |
| /cud(n) | Cursor down n lines (stop at bottom line) |
| /cuf(n) | Cursor forward n columns (stop at right margin). |
| /cup(l,c) | Position cursor at (line, column); origin is (1,1). |
| /cuu(n) | Cursor up n lines (stop at top line). |
| /dch(n) | Delete n characters |
| /dl(n) | Delete n lines |
| /ech(n) | Erase n characters (cursor does not move) |

| Mnemonic | Action |
|---|---|
| /ed0 | Erase from cursor to end of screen |
| /ed1 | Erase from beginning of screen to cursor |
| /ed2 | Erase complete screen |
| /el0 | Erase from cursor to end of line |
| /el1 | Erase from beginning of line to cursor |
| /el2 | Erase complete line |
| /eol | Erase to the end of line |
| /eos | Erase to the end of screen |
| /hoff | Highlight off |
| /hon | Highlight on |
| /ich(n) | Insert n blanks |
| /il(n) | Insert n lines |
| /ind(n) | Same as lf(n) |
| /lf(n) | Line feed n times (scrolls if necessary) |
| /ri(n) | Reverse line feeds n times (scrolls if necessary) |
| /roff | Reverse off |
| /ron | Reverse on |
| /sp(n) | Prints n blanks |
| /uoff | Underline off |
| /uon | Underline on |

# 4.2 Control Characters

According to the ANSI standard, the following control characters produce their usual effect:

| Character | ASCII | Action |
|---|---|---|
| BELL | 7 | Ring the bell |
| BS | 8 | Backspace 1 character |
| CR | 13 | Carriage return (go to column 0) |
| FF | 12 | Form feed (clear window) |
| LF | 10 | Line feed |

**Important:** All other control characters are ignored!

This means that literal control sequences (such as `W $c(27),"[...`) will not produce the expected behavior. All editing and formatting functions should be coded using the WRITE /<mnemonic> syntax.

# 5

# The Terminal Capabilities Database

The core of the character windows subsystem is completely terminal independent. At the lowest level, however, it needs to know many details about the current terminal. The system must know, for example, how many lines and columns the terminal has, how to position the cursor anywhere on the screen, and whether or not the cursor should automatically wrap to the next line when it reaches the last column.

This collection of information about a terminal is referred to as the "terminal capabilities". It is stored in the ^%SYS("tercap") global and loaded to memory when a /INIT command is issued. A terminal capabilities database contains the following items:

- Terminal names and aliases (e.g., "vt220|VT220|DEC-vt220")
- A set of boolean flags (e.g., has color, auto-wrap, etc.)
- A set of numeric constants (e.g., lines x columns)
- A set of control sequences (e.g., /cup, /el, etc.)
- A border map (tells which control sequences draw the borders)

An existing terminal description may be edited and new terminal descriptions may be created using the TERCAP utility.

## 5.1 The TERCAP Utility

TERCAP is the utility for creating/editing terminal descriptions. At most of the utilities prompts, you can see a list of options by entering ?. To run this utility from the %SYS namespace, enter:

```
%SYS>D ^TERCAP
```

Enter a *?* (question mark) at the Option prompt to display the three options:

```
Option: ?

     1 - Edit/create terminal description
     2 - Edit/create border map
     3 - Delete terminal description
```

- For details on option 1, see Creating and Editing Terminal Descriptions. This option allows you to define terminal names and aliases, set boolean flags, set numeric constants, and set control sequences.
- For details on option 2, see Creating and Editing Border Maps.
- Option 3 allows you to delete the description of a unwanted terminal.

# 5.2 Control Sequences

Control sequences sent to a terminal trigger some action at the terminal, like clearing the screen or positioning the cursor at a certain coordinate. Some sequences, like /clr, contain only literals (i.e., constant codes), while others, like /cup(l,c), also contain parameters. To ease the creation and maintenance of these tables, a special syntax resembling ObjectScript was devised. Generally, a control sequence may be a literal, an expression or a function. A literal is simply a number of ASCII characters inside double quotes (such as "[H").

If necessary, a quote symbol may be duplicated to be included as a literal. An expression is a parameter (%1, %2, and so on) optionally added to a decimal constant. For different mnemonics, the parameters mean different things. For the /cup(l,c), for example, %1 represents l (the line number) and %2 is c (the column number). In /ind(n), %1 is the repeat count for the "index" operation. Examples of expressions:

```
%1 %1+32 %2-1
```

The offset added to the parameter is limited to the range [-128,127].

A function is either $C(<list of codes>) or $A(<expression>). $C is just like the ObjectScript function of the same name. It is used to indicate control characters to be sent to the terminal, such as:

```
$c(27)
$c(155,31)
```

$A, however, is not the same as its ObjectScript counterpart. It indicates that the enclosed <expression> should be evaluated and sent to the terminal as ASCII characters and not in binary form.

`$c(65)` sends "A" (i.e., code 65)

`$a(65)` sends the string "65" (i.e., codes 54, 53)

`$a(%1)` sends %1 as a sequence of ASCII characters

`$a(%2+32)` evaluates %2+32 and sends it as a sequence of ASCII characters

For a complete example, consider the `/cup(l,c)` sequence for ANSI-like terminals:

```
$c(27),"[",$a(%1+1),";",$a(%2+1),"H"
```

Both parameters have 1 added to them before being sent, because their internal forms start at 0 and not at 1.

# 6

# Creating and Editing Terminal Descriptions

With TERCAP option 1 (see The TERCAP Utility), you can enter the following properties for a terminal:

- Terminal type name (or select from existing list)
- Number of lines and columns
- Boolean flags
- Mnemonics (entering *D deletes, after a confirmation, the source and binary forms of the current mnemonic).

## 6.1 Terminal Names and Aliases

When a new terminal type (name) is entered at the first prompt, the program allows you to copy the initial description from another terminal. This is useful if the description you are creating is similar to an existing terminal description.

You can check for other terminal types by entering a question mark at the Terminal type prompt:

```
DO tt^%CHARWIN
Terminal type: ?
  Terminal
  Terminal Wide Characters
  Linux
  SCO ANSI
  vt100
  vt220
  vt320
```

## 6.2 Boolean Flag Descriptions

The following flags can be used:

| Bit | Meaning |
|---|---|
| 1 | Terminal has color capability. |
| 2 | Terminal has windowing firmware (eg., "Waytec"). |
| 4 | Terminal is configured for auto-margin (or auto-wrap). |
| | Writing to the last column automatically positions the cursor at the first column of the next line. This depends on how the terminal is configured, either by a local menu or by the initialization string. The system works either way, but if this flag is of (like the default for the VT terminals), you may write at the lower right hand corner of the screen without scrolling it. If this flag is on, we never write to the last column of the last line. If you see a blank when another character should be there, it is the expected behavior. |
| 8 | Terminal has PC-like video attributes. |
| | For instance, it stores character attributes (blink and highlight) together with foreground and background colors in a single byte. When this flag is on, bit 1 should be off. Internally we store the attributes exactly like a PC video adapter card. This means that we spare the color byte without sacrificing the color capability. This results in less memory consumed and faster scrolls. This flag is intended for SCO UNIX® consoles (SCO ANSI and look alikes) and MS-DOS terminal emulators (Telix, Procomm, etc.). |
| 16 | The "erase character" operation |
| | Writes blanks with no attributes on. It has been noted that some implementations, like the SCO ANSI, maintain the current attribute and color when executing the "ech" function. Others, like the VT terminals, use a "normal" (i.e., with all attributes off) blank. This flag lets the system correctly use the "ech" operation (followed by "cursor forward") to write a long sequence of blanks without disturbing the screen attributes. |
| 32 | Disable border mapping. |
| | By default, the user must fill in a border map with the TERCAP utility. If the NLS output translation is being used, it may be desirable to perform the border mapping as part of this output translation. This flag thus disables the internal border mapping so that it can be done via the NLS output translation. |

# 6.3 Mnemonics and Their Parameters

Mnemonics may be entered via the TERCAP utility and are loaded by /INIT. In most cases, parameters may be omitted, and default values are assumed. For example:

```
WRITE /cup /sp /ed
```

is equivalent to:

```
WRITE /cup(1,1) /sp(1) /ed(0)
```

| Mnemonic | Description and parameters |
|---|---|
| aoff | All attributes off |
| bell | Ring the bell |

| Mnemonic | Description and parameters |
|---|---|
| boff | Blinking off |
| bon | Blinking on |
| clr | Clear the screen |
| coff | Cursor off (invisible) |
| con | Cursor on (visible) |
| cs01 | Switch from character set 0 to character set 1 |
| cs02 | Switch from character set 0 to character set 2 |
| cs10 | Switch from character set 1 to character set 0 |
| cs20 | Switch from character set 2 to character set 0 |
| cub(n) | Cursor backward n times |
| cud(n) | Cursor down n times |
| cuf(n) | Cursor forward n times |
| cup(l,c) | Position cursor at (line, column), origin = (0,0). |
| cuu(n) | Cursor up n times |
| dch(n) | Delete n characters |
| dl1 | Delete 1 line (idem) |
| dl(n) | Delete n lines |
| ech(n) | Erase n characters |
| ed0 | Erase from cursor to end of screen |
| ed1 | Erase from cursor to beginning of screen |
| ed2 | Erase complete screen |
| el0 | Erase from cursor to end of line |
| el1 | Erase from cursor to beginning of line |
| el2 | Erase complete line |
| end | Termination string |
| fill(l,t,r,b,c) | Fill a window with a character |
| | l = left column (from 0) |
| | t = top line (from 0) |
| | r = right column |
| | b = bottom line |
| | c = character code |
| hoff | Highlight off |
| hon | Highlight on |

| Mnemonic | Description and parameters |
|---|---|
| ich(n) | Insert n characters |
| il(n) | Insert n lines |
| il1 | Insert 1 line (useful if terminal does not have a generic "il") |
| ind(n) | Index n times |
| init | Initialization string |
| rep(c,n) | Repeat character c, n times |
| ri(n) | Reverse-index n times |
| roff | Reverse video off |
| ron | Reverse video on |
| uoff | Underline off |
| uon | Underline on |
| wbox(l,t,r,b,s) | Draw box |
| | l = left column (from 0) |
| | t = top line (from 0) |
| | r = right column |
| | b = bottom line |
| | s = border style (0=none, 1=single, 2=double) |
| wclose | Close window |
| wlimits(l,t,r,b) | Redefine window frame |
| | l = left column (from 0) |
| | t = top line (from 0) |
| | r = right column |
| | b = bottom line |
| wopen(l,t,r,b,s) | Open window (for terminals with windowing firmware) |
| | l = left column (from 0) |
| | t = top line (from 0) |
| | r = right column |
| | b = bottom line |
| | s = border style (0=none, 1=single, 2=double) |
| wuse(wid) | Make wid the current (top) window |

# 7

# Creating and Editing Border Maps

TERCAP Option 2 (see The TERCAP Utility), allows you to define a border map. Although the border map is part of a terminal description, it has a separate option purely for convenience. For each border style, the programmer can specify which control sequence should be sent to the terminal to draw each of the border characters.

A terminal is assumed to have a maximum of three different character sets or alphabets, from which to choose the border characters. The first character set (zero) is assumed to be the default, where ASCII (and possibly the whole Latin 1) is found. The other two (1 and 2) may be used for semi-graphical characters.

For each style, the user is prompted for the character set (0, 1 or 2) that includes the characters for that style and for the code in that character set that maps to each of the border characters. The control sequences that select among the three character sets are entered as if they were mnemonics at option 1:

| Mnemonic | Original Character Set | Resulting Character Set |
|---|---|---|
| cs01 | 0 | 1 |
| cs02 | 0 | 2 |
| cs10 | 1 | 0 |
| cs20 | 2 | 0 |

*Example:*

Consider the case of the Digital Equipment Corporation (DEC) VT320 terminal. During the /INIT command, you set the terminal to use the DEC Special Graphics Character Set (the one that contains the border characters) as G1. G0 is the default Latin 1. To switch from G0 to G1, you send a SO (shift out = ASCII 14) control and to switch from G1 to G0 a SI (shift in = ASCII 15). Therefore, the following definitions apply:

```
cs01 = $c(14)
cs10 = $c(15)
```

The upper left corner (UL) character occupies position 108 in the DEC Special Graphics Character Set. This is the code that must be supplied to the TERCAP utility. Caché then takes care of switching between the two character sets and optimizing the control sequences that are actually sent to the terminal. If two or more border characters occupy contiguous positions in a line (such as in the top and bottom borders), then a single SO is sent in the beginning, followed by the border codes in G1, ended by a single SI.

In general, cs01, cs02, cs10 and cs20 may be multi-byte character sequences.

# 7.1 Border Characters and Codes

Caché can be installed in either 8-bit or 16-bit (Unicode) mode. The 8-bit mode supports the display of the ISO Latin character sets while the 16-bit mode uses the Unicode 2.0 character set. These sets are used to store all the characters internally in globals, local variables and also in the window structures. By default, the Unicode collation is used for storage. External representations may be converted to the internal character set by I/O translation tables.

These internal character sets do not include the semi-graphical characters needed for drawing window borders. Arbitrary codes were chosen to internally represent border characters because they will be later converted to their corresponding external representations.

The table below and the TERCAP utility use the following abbreviations:

- UL — upper left corner
- UR — upper right corner
- LL — lower left corner
- LR — lower right corner
- TO — top border
- BO — bottom border
- LE — left border
- RI — right border

Generally, these eight characters are necessary to draw a complete window border. However, in practice, they can be reduced to six, four or even one, depending on the terminal at use and on the border style.

For those who want to provide their own output translation tables, the following list of internal codes for borders may be useful.

| Style | Character | Code |
|---|---|---|
| 1 | BO | 140 |
| 1 | LE | 138 |
| 1 | LL | 132 |
| 1 | LR | 134 |
| 1 | RI | 142 |
| 1 | TO | 136 |
| 1 | UL | 128 |
| 1 | UR | 130 |
| 2 | BO | 141 |
| 2 | LE | 139 |
| 2 | LL | 133 |
| 2 | LR | 135 |
| 2 | RI | 143 |

| Style | Character | Code |
|---|---|---|
| 2 | TO | 137 |
| 2 | UL | 129 |
| 2 | UR | 131 |
| 3 | — | Style 3 (bold line) is like style 1 (single line) with the "bold" attribute forced on. |
| 4 | ALL | 158 |
| 5 | ALL | 157 |
| 6 | ALL | 156 |
| 7 | BO | 149 |
| 7 | LE | 150 |
| 7 | LL | 146 |
| 7 | LL | 147 |
| 7 | RI | 151 |
| 7 | TO | 148 |
| 7 | UL | 144 |
| 7 | UR | 145 |
| 8 | — | Style 8 (medium bar) is the same as style 7 (light bar). |
| 9 | BO | 155 |
| 9 | TO | 154 |
| 9 | UL,LE,LL | 152 |
| 9 | UR,RI,LR | 153 |