APPENDIX

A

Routine Development in Programmer Mode

This appendix describes ways of creating and editing intermediate code routines that are alternatives to the Caché Studio editor.

With the methods described here you can create only pure Caché ObjectScript source code.

Creating and Editing Intermediate Code Routines

You create intermediate code routines by:

- Entering code in the Caché Studio editor. You create intermediate code or macro code in this way.
- Compiling macro source routines already entered in the Caché Studio.
- Developing code at the Caché Programmer Mode prompt. You create pure Caché ObjectScript source code in this way.

Intermediate code routines can be edited with the:

- Caché Studio — see Chapter 5, “Caché Studio.”
- Routine Line Editor — see “Routine Line Editor” on page A-10.

Managing Routines at the Caché Programmer Prompt

To write pure Caché ObjectScript routines, sign onto Caché in Programmer Mode. For example, use the Caché Terminal.

When you enter Programmer Mode, Caché displays the current namespace and angle bracket prompt (>), which indicates that you can now type Caché ObjectScript commands. Every time the prompt appears you can type one or more commands. When you press <RETURN>, the commands are executed immediately.

Before displaying the (>) prompt when you enter Programmer Mode, Caché makes your principal device the current device, as though a USE 0 had been executed.

To exit Caché and return to the host operating system, use the Caché command HALT.

```
USER>HALT
```

Creating a New Routine

1. Use the NEW command at the Caché Programmer prompt:

   ```
   USER>NEW
   ```

2. To enter Caché ObjectScript command lines into the routine, press the <TAB> key once, type the command, and press the <ENTER> key. For example:

   ```
   USER 1S1>READ NUM1,NUM2,NUM3
   USER 1S1>SET X = NUM1+NUM2+NUM3
   USER 1S1>WRITE !,"The sum is ",X
   USER 1S1>NEW
   ```

3. Save the routine you have entered to the CACHE.DAT file on disk in the current namespace by using the ZSAVE command with the routine name:

   ```
   USER>ZS ADDNUM
   ```

Caché saves both the intermediate code versions of the routine.

Loading a Routine with ZLOAD

To load an existing routine into your user process space, use the ZLOAD command. Once loaded, a routine remains the current one for your process until you load another routine explicitly, with a ZLOAD command, or implicitly, with a DO or a GOTO command. As long as your routine is current, you can print or edit it.

The format of the ZLOAD command is:

ZLOAD [*routine_name*]

Loading an Existing Routine

To load a routine in the current namespace, enter ZLOAD followed by the routine name.

```
USER>ZL ROUT
```

Caché looks in the pool of routine buffers in memory for the routine and, if it is not already there, loads the routine object code into one of the buffers. The source code remains in ^ROUTINE. Some implementations do not have a routine buffer pool, in which case the compiledcode is loaded directly into the user's process private space rather than mapped in a common memory routine buffer.

Loading a Routine from the Current Device

Use the ZLOAD command without an argument to load lines of code from your current device (that is, the device to which I/O is currently directed). Line loading will continue until the system reads a null-string line. This loaded routine has no name until you file it with the ZSAVE command.

Printing Lines

To display a routine that has been loaded into your user process space, use the PRINT (P) command, which has several forms. Keep in mind that in order to print a routine, you must have the privilege to read the ^ROUTINE global in which it resides, and its intermediate code version must be there. If Caché displays only empty lines when you attempt to PRINT a routine, either the routine has no intermediate code version or you lack sufficient privilege to read it.

To display an entire routine, enter PRINT without an argument, as in this example:

```
USER>ZL ROUT
USER>P
ROUT    SET END=^ACC(ID,0)
        FOR J=1:1:END SET X=^ACC(ID,J) DO ^XYZ
        SET X=3
        QUIT
USER>
```

To display a specific line, enter PRINT followed by a tag, a tag+offset, or a +offset:

```
USER>P ROUT+2
SET X=3
USER>
```

To display a range of lines between tag A and tag B inclusive, enter PRINT in the following format:

```
USER>P A:B
```

In response to the following code, lines A+3 through B+5 are printed:

```
USER>P A+3:B+5
```

To print routine lines on another device, you must direct the I/O to that device by typing the USE command.

In response to the following code, the routine ROUT is printed on the console window:

```
USER>ZL ROUT ; load routine ROUT
USER>OPEN 1 USE 1 P ; open and use the console, print
USER>CLOSE 1 ;close the device
```

Saving Routines

When you are writing or editing a routine, you are working on a copy of that routine. Use the ZSAVE (ZS) command to save any changes. ZSAVE stores the version of the routine in your user process to disk under the default namespace.

You can use ZSAVE with or without an argument. If you enter ZSAVE followed by a routine name, as in:

```
USER>ZS ROUT
```

Caché stores the routine under the name specified (ROUT, in this example). When you enter ZSAVE without an argument, as in:

```
USER>ZS
```

Caché saves the routine under the same name as the original (more exactly, the name in the Caché variable $ZN), replacing any earlier version on the disk. Thus the system irretrievably deletes the old version of a routine and saves only the newest version.

If you make a mistake while editing or do not want to save the routine, you do not need to ZSAVE the routine. Instead, ZLOAD that routine or any other, or DO another routine, and those changes disappear.

Inserting Lines

You insert lines of code in an Caché routine from Programmer Mode by:

- Entering a line containing a <TAB> character.
- Using the ZINSERT command.

You can also use the following tools to insert lines:

- Caché Studio.
- Routine Line Editor. Note that this tool does not operate on macro source code or include files.

Inserted lines change only a copy of the routine. You must use the ZSAVE command to save the modified routine on the disk.

Edit Pointer

The edit pointer indicates where insertions will be placed in the routine by pointing to the place where the line will be inserted.

When a routine is loaded, the edit pointer is positioned at the beginning of the routine.

- The PRINT command leaves the pointer immediately after the last line printed.
- When an insertion is made, the edit pointer is left after the inserted line.
- When a deletion is made, the edit pointer is left at the place vacated by the deleted lines.

Inserting a New Line of Code With the Tab Key

To insert a new line of code, type the tag (if any), <TAB> (or <CONTROL-I> on terminals without a tab key), and the body of the line containing the (intermediate code) commands. If the line has no tag, begin with <TAB>. Caché then inserts the line at the place designated by the edit pointer. Caché translates the <TAB> into a space character in the routine line. The <TAB> character in your typed line tells Caché to insert the line as a routine line rather than execute it immediately.

Note: You cannot use this technique to insert a line at the beginning of a routine line; there, you must use the ZINSERT command.

The following example shows how to insert a line of code into a specific location within a routine. In the example, the ZLOAD command loads the routine ROUT and the PRINT command is used to display it on your terminal. A line of code ("B SET X=3") is then inserted into the routine after the line ROUT+1. There are two steps to that code insertion:

1. Print the line before the one to be inserted in order to set the edit pointer.
2. Enter the new line with a <TAB> after the tag B. The routine is then displayed, showing the inserted line at tag B.

```
USER>ZL ROUT
USER>P
ROUT    SET END=^ACC(ID,0)
        FOR J=1:1:END SET X=^ACC(ID,J) DO ^XYZ
        QUIT
USER>P ROUT+1
        FOR J=1:1:END SET X=^ACC(ID,J) DO ^XYZ
USER>B SET X=3 ; this line has a tab after the B
USER>P
        ROUT  SET END=^ACC(ID,0)
        FOR J=1:1:END SET X=^ACC(ID,J) DO ^XYZ
B       SET X=3
        QUIT
```

Inserting a New Line of Code With ZINSERT

The ZINSERT (ZI) command is used to insert a single line of code into a routine at the edit pointer or at a specific location within the routine.

The ZINSERT command is used by editors; a programmer would not normally use it to type a line of code directly into a routine.

To insert a line at the edit pointer, type the ZI command in the following format:

ZI *code*

where *code* is a string consisting of an optional tag, a space, and a line of code.

For example, the following expressions would both be inserted after the edit pointer:

```
USER>ZI " SET X=3"
USER>ZI "XYZ SET X=3"
```

To insert a line at another location, type the ZI command using the following format:

ZI *code:location*

where *location* is a tag or tag+offset. For example:

```
USER>ZI "ABC SET A=2":DEF+2
```

You may also simply use +*offset*, where *offset* is the absolute number of a line in the routine. Use +0 to insert a line at the beginning of the routine. Do not include a <TAB> in *code* or the entire line gets put in the buffer.

This example shows how to use the ZINSERT command to insert a line of code at a specific location within a routine. The ZLOAD command loads the routine ROUT. The PRINT command then displays the routine on the terminal and a line of code with tag B ("B SET X=3") is inserted into the routine, after line ROUT+1. Finally, the routine is displayed, showing the inserted line at tag B.

```
USER>ZL ROUT
USER>P
ROUT    SET END=^ACC(ID,0)
        FOR J=1:1:END SET X=^ACC(ID,J) DO ^XYZ
        QUIT
USER>ZI "B SET X=3":ROUT+1
USER>P
ROUT    SET END=^ACC(ID,0)
        FOR J=1:1:END SET X=^ACC(ID,J) DO ^XYZ
B       SET X=3
        QUIT
USER>
```

Removing Lines of Code

Use the ZREMOVE (ZR) command to erase a single line or range of lines from the current routine, or to erase the entire current routine.

The ZREMOVE command does not delete a routine or any part of one from the disk; it affects only your copy of the routine. You must use the ZSAVE command to save the modified routine or the %RDELETE utility to delete the entire routine from the disk.

If you use ZREMOVE without an argument, Caché erases all of the lines from your local copy of the routine, as in this example:

```
USER>ZL ROUT
USER>P
ROUT    SET END=^ACC(ID,0)
        FOR J=1:1:END SET X=^ACC(ID,J) DO ^XYZ
        QUIT
USER>ZR
USER>P
USER>
```

If you type a command statement with the format:

```
USER>ZR A
```

where ***A*** is a tag, a tag+offset, or simply +offset, the system erases the designated line from the routine.

Entering a command statement with the format:

```
USER>ZR A:B
```

erases all lines from ***A*** to ***B*** inclusive, as in:

```
USER>ZR X+4:Y+3
```

The following example shows how to erase a group of lines by using the ZREMOVE command. First, the ZLOAD command loads the routine ROUT and the PRINT command displays it. Next, the lines indicated by ROUT+1 through ROUT+2 are removed, using the command statement ZR ROUT+1:ROUT+2. Finally, the routine is printed to illustrate how the two lines have been erased.

```
USER>ZL ROUT
USER>P
ROUT    SET END=^ACC(ID,0)
        FOR J=1:1:END SET X=^ACC(ID,J) DO ^XYZ
        SET X=3
        QUIT
USER>ZR ROUT+1:ROUT+2
USER>P
ROUT    SET END=^ACC(ID,0)
        QUIT
USER>
```

Deleting an Entire Routine

To delete an entire intermediate code routine that is stored on disk, enter an argumentless ZREMOVE command and then a ZSAVE (ZS) command.

To remove a routine named MYROUT, enter the following:

```
USER>ZR
USER>ZS MYROUT
```

You can also delete routines from disk by using the %RDELETE utility or the Caché Explorer.

Modifying Routines Without Source Code

If you are running Caché routines present only in m-code versions, or if the source code in the global ^ROUTINE has been deleted, you can still use Caché commands to modify these routines. ZLOAD, ZREMOVE, ZINSERT, and ZSAVE still work, as does entering code in Programmer Mode with a <TAB>. You cannot see the former source code, so the Caché Routine Line Editor is not useful for editing in this situation. Caché stores these new lines of source code in the ^ROUTINE global, forming an audit trail of the inserted lines.

The following command sequence shows how you can change a line of code even though the source code version of the routine is not present.

```
USER>ZL ROUT
USER>ZR X+5
USER>SET Y=Z*2; a tab before the S
USER>B    IF Y<X DO ^ABC ; a tab after the B
USER>ZS
```

In this example, routine ROUT, which exists only in m-code, is loaded into the routine storage area. The ZREMOVE command deletes line X+5, and two new lines replace it. Finally, the ZSAVE command is used to store the new m-code version of the routine.

If you attempt to PRINT a line whose source code has been deleted a blank line will be displayed. A PRINT of a routine whose source has been deleted will result in the display of only those lines that have been added since the source was deleted.

Routine Line Editor

The Caché Routine Line Editor allows you to edit Caché ObjectScript source code lines. The Routine Line Editor performs both routine and global editing, so you can modify global nodes as well as intermediate code lines. The Routine Line Editor is intended for editing and debugging of pure Caché ObjectScript routines. It does not operate on macro source routines or include files.

Routine line editing allows you to:

- Edit lines
- Insert new lines
- Perform search and replace operations on character strings

Invoking the Routine Line Editor

To use the Routine Line Editor, you must enter Programmer Mode. First, load the routine you want to edit using the ZLOAD command:

```
USER>ZLOAD ROUT
```

Then, enter:

```
USER>X ^%
```

Caché responds with:

```
Edit:
```

In response to the "Edit:" prompt, specify one of the editing features listed below and press <RETURN>. You can also enter a ? (question mark) for a list of valid commands.

An invalid response may cause an Caché ObjectScript language error and return you to the Programmer Mode prompt (>). Usually, however, the Line Editor can recognize an invalid response, in which case it displays "???" and returns to the "Edit:" prompt. If an error should return you to the Programmer Mode prompt, you can reinvoke the Line Editor by typing:

```
X ^%
```

The following table describes the routine line editing features:

Feature	Function
Tag (or tag+offset or tag-offset)	Edits the line specified by that tag.
<return>	Re-edits the last line edited, or inserted, if any.
+	Edits the line immediately following the last edited line.
-	Edits the line immediately prior to the last edited line.
+n	Edits the *n*th line after the last edited line. (e.g. +3).
-n	Edits the *n*th line before the last edited line. (e.g. -3).
""+n	Edits the *n*th line in the routine. (e.g., ""+3 is the third line).
.	Returns to Programmer Mode from the Routine Line Editor.
.B	Breaks a line into two lines.
.C	Changes all occurrences of one string to another string within a specified group of lines.
.F	Files the current routine under the current routine name. The current date and time are added as a comment in the text of the first routine line. This feature may affect any other process that is running the routine.
.E	Not used.
.I	Inserts one or more lines after another line.
.J	Joins two consecutive lines into one.
.M	Moves lines of code from one place to another in the same routine.
.R	Removes a range of lines from a routine.
.S	Searches for all occurrences of a string in the routine.

Feature	Function
Global reference	Allows you to edit a global node by entering a full global reference.
^	Re-edits the previously edited global node.
?	Displays a list of Routine Line Editor commands.
Caché ObjectScript	Allows you to execute most lines of Caché ObjectScript code from within the Routine Line Editor.

Executing ObjectScript Code in the Routine Line Editor

If you enter a line of Caché ObjectScript code at the "Edit:" prompt, the Routine Line Editor will execute that line; for example:

```
Edit:WRITE 285.67-49.81
235.86
Edit:
```

However, you should be aware of the following limitations:

- The line of Caché ObjectScript code must contain at least one space character. If you only want to execute an argumentless command (such as PRINT or WRITE), follow it with a space.
- Calling any routine other than the one you are currently editing before you refile the edited routine nullifies any changes made since the routine was last filed.
- The Line Editor itself uses local variables that may be affected by Programmer Mode commands. These variables all start with a percent sign. One of these variables, %NX, controls branching within the Line Editor. If %NX is killed, the routine returns to Programmer Mode. Thus entering K after the "Edit:" prompt destroys all local variables and returns you to Programmer Mode.
- Do not issue a ZLOAD command from within the Line Editor; the Line Editor uses the XECUTE command. If you try to load a new routine while you are in the Line Editor, the system will reload the old routine at the end of the XECUTE. To edit a routine, you must exit the Line Editor, load the routine with the ZLOAD command in Programmer Mode, and then reenter the Line Editor with by typing "X ^%".

Saving Routines and Exiting

Save edited routines by using either the .F command of the Line Editor or the ZSAVE command in Programmer Mode. Do not use the ZSAVE command in

response to the Edit: prompt. Remember that the .F command of the Routine Line Editor adds a comment to the first line of the routine-the date and time of editing.

Exit the Routine Line Editor by typing a period (.) at the Edit: prompt.

Editing a Single Line

Video Terminal (Including Windows Terminal Emulator)

In response to the Edit: prompt, choose the line to be edited, as previously described. Caché displays this line. If there is a tag, a single space character separates it from the main body of the routine line. Next, the cursor is positioned over the first character, and you can begin editing.

First, position the cursor in any of the following ways:

- To move the cursor forward to the next space or punctuation mark, press the space bar. (The space bar moves you through a series of spaces or punctuation marks one at a time.)
- To move the cursor forward one character position, type a period (.).
- To move the cursor back one character position, press <BACKSPACE>.

When the cursor is properly positioned, you can delete code, enter code, or return the cursor to the beginning of the line, as follows:

- To delete the single character marked by the cursor, press <DELETE> or <RUBOUT>.
- To delete a series of characters, starting with the current character and ending with the character before the next space or punctuation mark, type D (for delete).
- To enter one or more characters, type E (for enter), type the new characters, and press <RETURN>.
- To return the cursor to the beginning of the line, press <RETURN>. Pressing <RETURN> while the cursor is at the beginning of the line returns the cursor to the Edit: prompt.
- After making an edit, the cursor remains in its last position. You then can move the cursor to a new position to perform another edit.

This example shows you how to replace the code **Y=2** with the code **Z=A>5**. The caret character (^) represents the cursor. You must select a line first and then

position the cursor. After you select a line, the information appears on the screen as follows:

```
Edit: TAG
TAG S X=3,Y=2 D ABC
^
```

The cursor is positioned at the T in TAG. Pressing the space bar once moves the cursor to the first space:

```
TAG S X=3,Y=2 D ABC
^
```

Pressing the space bar twice more moves the cursor to the next space and then to the comma:

```
TAG S X=3,Y=2 D ABC
^
```

Typing a period moves the cursor forward one position:

```
TAG S X=3,Y=2 D ABC
^
```

To delete the code **Y=2**, type a D (for delete) and the screen then appears as follows:

```
TAG S X=3 D ABC
^
```

To insert the new code Z=A>5, type an E (for enter). No change appears on the screen until you begin typing in new characters. You must press <RETURN> to stop entering new characters. Type the characters Z=A>5, press <RETURN>, and the screen will look like this:

```
TAG S X=3,Z=A>5 D ABC
^
```

You can now reposition the cursor for another edit. For example, if you want to change the code X=3 to X=2, press <BACKSPACE> seven times to position the cursor over the 3, press <DELETE> once to delete the 3, type E for enter, type 2, and press <RETURN>. To move the cursor back to the first character in the line, press <RETURN> again. To return to the Edit: prompt, press <RETURN> while the cursor is at the first character.

Printer Terminal

Hard copy (printer) terminals require a different technique for line editing, which you may also prefer when you use a video terminal at a low baud rate, such as 300 baud. (To force a terminal to act like a printer, set variable XY to null before you call the Line Editor).

In response to the Edit: prompt, select the line to be edited, as previously described. An R (for replace) appears. Then enter one of the following responses:

- To print the current line and return to the Edit: prompt, press <RETURN>.
- To replace a string of characters in a line with another string, specify the characters to be replaced.
- To replace all characters between two strings, enter the first string (S1) followed by three periods and the second string (S2):
  S1...S2
- In the previous operation, if you enter the first string, three periods, and no second string (S1...), you can replace S1 and all subsequent characters in the line. Similarly, by entering the three periods and then the string (...S2), you can replace S2 and all characters preceding it in the line.
- To add a string to the end of the line, type END.

After you enter one of these responses, the Line Editor gives you the prompt W (with). Simply enter the new string and press <RETURN>. When you finish, the computer will prompt again with an R for another replace. You can make another edit, or press <RETURN> in response to R to print the edited line and return to the Edit: question.

For example, suppose the following routine is in your routine buffer:

```
SAMP ;SAMPLE ROUTINE TO BE EDITED
    WRITE "THIS IS THE SECOND OUTPUT TINE",!
    HALT
```

You can edit it as follows:

```
USER>X ^%
Edit: SAMP+1 READ SECOND WRITE FIRST READ <RETURN>
WRITE "THIS IS THE FIRST OUTPUT TINE",!
Edit: -1 SAMP READ TINE... WRITE LINE READ <RETURN>
SAMP ; SAMPLE ROUTINE <RETURN>
Edit: SAMP READ END WRITE TEST READ <return>
SAMP ; SAMPLE ROUTINE TEST
Edit: .
USER>
```

The routine now reads:

```
SAMP; SAMPLE ROUTINE TEST
    W "THIS IS THE FIRST OUTPUT LINE",!
    HALT
```

Routine Line Editor Commands

You can modify routines and global data on CRTs by using the commands listed below. The list includes the type of change, what you must enter in response to the Edit: prompt, and the name of the command. After typing the period and the command letter, press <RETURN> to invoke the command.

Modification	Your Entry	Command Name
Split a line	.B	BREAK
Replace a string	.C	CHANGE EVERY
File a routine	.F	FILE
Insert a line	.I	INSERT
Combine two lines	.J	Join
Move a line	.M	MOVE
Erase a line	.R	REMOVE
Search for a string	.S	SEARCH

All the commands described so far treat a line tag as part of the line. The delimiter between a tag and the rest of the line is a space. Be careful that you do not accidentally modify a tag when you intend to modify another part of the line. If you try changing the tag to an invalid format, the Line Editor allows you to re-edit the line to correct the syntax.

BREAK

To split one line into two, use the BREAK command. The Line Editor asks for the line to break and where the break should be inserted in the line. The default is the current line.

```
Edit: .Break line: ROU
After characters: Z=3
```

CHANGE EVERY

When you issue the CHANGE EVERY command, the computer prompts with the following:

```
Change Every:
```

Enter the string you want to replace. (A null response returns you to the Edit: prompt.) The computer then prompts with:

```
to:
```

Enter the replacement string. If you press <RETURN> the occurrence of the string to be replaced will be deleted. The computer next asks:

```
From line:BEG=>
```

Respond with the tag, or with the tag+offset, where you want the change. All preceding lines will be unaffected. The default, BEG, indicates the first line of the routine.

The Routine Line Editor now asks for the last line to change. No line after that line will be affected. The default, END, indicates the last line of the routine. The Line Editor makes the changes and prints the new lines.

For example:

```
Edit: .Change every: S A= to: S B=
From line: BEG=> TG+1 to line: END=> B+4
TG+4 S B=100,C=99
B+1 W X S B=B+1
Edit:
```

The Routine Line Editor has changed all occurrences of S A= to S B=, from lines TG+1 through B+4, and lists the two new lines.

FILE

The FILE command files the current routine to disk under its current routine name. Before filing the routine, the FILE command enters the present date and time as a comment in the first line of the routine. This command may affect any other process that is running the routine.

```
Edit: .File
Edit:
```

INSERT

Use the INSERT command to insert routine lines directly from the Routine Line Editor. After you enter .I, the computer asks:

```
Insert after:
```

Specify the line after which you want to insert the new lines. The current line is the default. (If you do not specify a line currently in the routine, the Line Editor returns to the Edit: prompt.) The computer now asks:

```
Line:
```

Respond by entering the new line of the routine in the following format:

[*optional_tag*][space][*text_of_line*]

If you enter a line with an illegal tag, you get a <Tag Syntax> error and the line is not stored. Note that the delimiter between the tag and the text of the line is a space, not a tab. The Line Editor continues with the Line: prompt until you enter a null response, at which point it returns to the Edit: prompt.

```
Edit: .Insert after: loop+1
Line: S X=1,Y=2
Line: ABC W "NAME:",!
Line:
Edit:
```

JOIN

Use the JOIN command to combine two consecutive lines into one. The Line Editor asks for which line to join; the specified line is joined to the line immediately following it. The defaults are the current line and the line immediately following it.

```
Edit: .Join line: ROU and ROU+1
ROU s x=1,y=2,z=3 d ABCMOVE
```

MOVE

The MOVE command allows you to move lines of code from one place to another in the same routine; the system erases the lines from the original location. The Line Editor asks you for the first and last lines to move as well as where to move them. After moving the lines, the Line Editor prints the entire routine.

```
Edit: ROU
ROU s x=1,y=2,z=3 d
Edit: .M
Begin: ROU+2 End: ROU+3
Insert After: ROU+4
ROU s x=1,y=2,z=3
    r sum
    i z>sum w sum
    i sum>z w z
    q
```

REMOVE

Use the REMOVE command to erase a range of lines in a routine.

```
Edit: .Remove lines; from line: BEG=> ROU+2
to line: END=> ROU+3
OK to remove lines? y ...deleted lines
```

SEARCH

The SEARCH command works like the CHANGE command, but no replacement string is specified and no changes are made to the routine.

```
Edit: .Search for: S A=
From line: BEG=> TG+1 to line: END=> B+4
TG+4 S A=100,C=99
B+1 W X S A=B+1
Edit:
```