Article
Leo Makowski · Jul 15, 2016 5m read

Zinsert and friends: Coding in terminal

While Studio and Atelier are useful development interfaces, there are occasionally situations where a quick edit needs to be made to code and only terminal access is available. A useful set of tools to do this are the zload, zprint, zinsert, zremove, and zsave commands. These are abbreviated to zl, zp, zi, zr, and zs respectively. While each of these commands has its own page in documentation, this article will synthesize that information with examples to provide instruction for their combined use.

Zload – loads a Caché ObjectScript INT routine into the routine buffer

Usage: >zl routinename

>zl

Zload with an argument is passed a routine name without a preceding “ ^ ” . This will look for an INT routine with that name in the current namespace and attempt to load it into the routine buffer. If no INT routine with that name can be found then a <NOROUTINE> error will be thrown. An argumentless zload can be used programmatically to load the contents of the current device (such as a text file) into the routine buffer. Using an argumentless zload in the terminal will allow the user to write to the routine buffer directly. A new line can be started with <enter>, and any line of code not preceded by a space or <tab> will be treated as a label. Pressing <enter> twice will commit the entered text to the buffer. Using zload when there is already code in the routine buffer will overwrite the current contents.

Zprint – displays the contents of the routine buffer on the current device

Usage: >zp

>zp offset

>zp offset1:offset2

Zprint can be called argumentless or can take as arguments an offset or range of offsets. Zprint with no arguments will print every line of code in the current routine buffer. If the command is given a single offset, it will print a specified line of code. For example, >zp +2 would print the second line of code in the routine buffer. When given two offsets, a range of code is printed; >zp +5:+7 means print lines 5, 6, and 7. Offsets can be for the entire routine or with reference to a label. Keep in mind that the first line of code (offset +1) in any routine will usually be the routine name.

Zinsert – insert a line of code after an offset in the routine buffer

Usage: >zi “ code ”

>zi “ code ” :offset

The “ code ” passed as an argument to zinsert should be a string. If no offset is specified, then the “ code ” will be appended to the end of the current contents of the routine buffer. Otherwise, the “ code ” will be added after the offset specified. E.g. >zi “ code ” :section+1 will insert “ code ” after the first line of the “ section ” subroutine, and the inserted code can then be referenced at the section+2 offset. The string passed as an argument should start with a space (“ set ^x=3 ”) unless the user is adding a label to the code. The preceding space will be treated as a <tab> and code passed this way will be executable.

Zremove – remove a line of code at an offset in the routine buffer

Usage: >zr

>zr offset

>zr offset1:offset2

An argumentless zremove will clear out the contents of the routine buffer. If an offset or range of offsets is specified, then just those lines of code will be removed from the routine buffer.

Zsave – saves the contents of the routine buffer to a database (as INT code)

Usage: >zs

>zs routinename

If the contents of the routine buffer were zloaded from an existing routine in a database (i.e. >zload routinename) then an argumentless zsave will save the modified buffer as INT code for that routine. Otherwise, an argumentless zsave will generate a <COMMAND> error. Zsave can take a routine name as an argument, and will save the contents of the buffer as INT code for that routine name (i.e. routinename.INT) in the current namespace. For example, a routine saved with >zs myroutine can then be accessed by calling ^myroutine. Because the buffer is saved into INT code, any recompile of the MAC code of the associated routine will overwrite the changes.

Example

Given the following routine:

```
1 testroutine
2 do tag1
3 do tag2
4 quit
5 tag1
6 set ^x(1)="foo"
7 return
8 tag2
9 set ^y(1)="superman"
10 return
11 tag3
12 set ^z(1)="blue"
13 return
```

```
 USER>d ^testroutine

USER>zw ^x

^x(1)="foo"

USER>zw ^z

USER>zl testroutine

USER>zp +2:+3

        do tag1

        do tag2

USER>zi " do tag3":+3
USER>zi " set ^x(2)=""bar""":tag1+1
USER>zr tag3+1
USER>zi " set ^z(1)=4":tag3
USER>zp
testroutine
        do tag1
        do tag2
        do tag3
        quit
tag1
        set ^x(1)="foo"
        set ^x(2)="bar"
        return
tag2
        set ^y(1)="superman"
        return
tag3
        set ^z(1)=4
        return
USER>zs
USER>d ^testroutine
USER>zw ^x
^x(1)="foo"
^x(2)="bar"
USER>zw ^z
^z(1)=4
USER>zr
USER>zp
USER>
```

Because the changes made are saved into INT code, this method of editing is not ideal for making robust changes. There are still times when quick, temporary changes need to be made and these tools work best for that purpose.

#Development Environment #Terminal #ObjectScript

Source URL:https://community.intersystems.com/post/zinsert-and-friends-coding-terminal