Article
Steve Pisani · Jun 6, 2016 7m read

Language Extensions

This is a posting about a particular feature of Caché which I find useful but is probably not well known or used. I am referring to the feature of Language Extensions.

This feature allows you to extend the commands, special variables and functions available in Caché Object Script with commands, special variables and functions of your own. This functionality also applies to other languages the Caché supports at the server, including Caché Basic and Multivalue Basic.

Why would I need or want to add new commands ?

There are a number of use cases. Typically, I like to use this to save time during development and debugging, by wrapping up frequently used function calls, or a series of commands, or just a long function call, into a single abbreviated custom command.

Let's look at an example. Developers debugging at the command line regularly need to retrieve the error text of the last error raised, represented by the status object variable (%objlasterror), that would be defined in their partition. This is how you do it:

```
> Do ##class(%SYSTEM.Status).DisplayError(%objlasterror)
```

In the above call, we are invoking the DisplayError method from the %SYSTEM.Status class, passing in %objlasterror. This is a long string to type frequently, inviting typos causing delay. What we can do is introduce our own site ’ s short command, that would do the same thing. Our new command could be designed to optionally take any status object variable as an argument, and left off, %objlasterror is assumed to be the argument.

Here is the new command which I have called ‘ ZOE ’ , in action:

```
SAMPLES>set p=##class(Sample.Person).%New()
SAMPLES>do p.%Save()   ; (save will fail, and define %objlasterror).
SAMPLES>zoe
ERROR #7209: Datatype value '' does not match PATTERN '3N1"-"2N1"-"4N'
```

```
SAMPLES>set tSC= ##class(%SYSTEM.Status).Error(5001, ” Sample Status Text ” )
SAMPLES>zoe tSC
ERROR #5001: Sample Status Text
```

So how are custom commands, special variable and functions implemented ?

Extensions are implemented by creating a Caché Objectscript Routine specifically called %ZLANGCnn, %ZLANGVnn or %ZLANGFnn, that define the logic behind new commands, special variables and functions respectively.

The nn in the above routine names, denote the languages in which these extensions are to be made available.

Common nn values include:

00 - Caché Object Script
09 - Caché Basic
11 - Multivalued Basic

(See documentation link at the bottom of this post for the full list)

I have built the routine %ZLANGC00 to define ZOE command above, and saved this in the %SYS namespace. I can add as many language extension commands as a I like inside of the same routine (and that routine can of course invoke other code).

Here is a sample %ZLANGC00 routine I defined for this post:

```
%ZLANGC00 ; Language Extensions (Commands) - SP
  ;
  Quit   ; quit nicely if this routine is actually invoked directly.

   ; Command to display error text of status object supplied, OR of %objlasterror
  ; if no argument defined.
ZOERROR(%err) do ZOE(.%err) ; note the '.' so it works if %err is undefined.
ZOE(%err) ;
  if $get(%err)="",$get(%objlasterror)'="" {
  set %err=%objlasterror
  }
  if $get(%err)'="" do $System.OBJ.DisplayError(%err)
  quit

ZOLIST ; Command to quickly output the list of defined objects in my partition
ZOL ;
  do $System.OBJ.ShowObjects()
  quit
```

Notice also that in the above example, that I have also introduce a long-handed alias (ZOERROR) for the ZOE command which will drop through and do exactly the same thing. There is also a second, argument-less command ZOL (ZOLIST) for dumping out the currently defined object variables. Note that when implementations of custom code do not take arguments (eg ZOLIST, ZOL) code can just drop through from one line label (ZOLIST) to the next. However, if arguments are required (eg ZOERROR and ZOE), then the long-handed function needs to explicitly call the short-handed function.

Similarly, you can create your own custom Functions. Here is the implementation of a function to return the r , instead of commands. Consider for example, a function to return Year Quarter given a $Horolog date.

```
%ZLANGF00 ; Language Extensions (Functions) - SP
  ;
  Quit   ; quit nicely if this routine is actually invoked directly.

  ; Function to return the quarter (Q1, Q2, etc..) of a given date supplied in the
  ; internal date format ($HOROLOG).
  ;
ZYEARQ(hdate)   ;
  do ZYQ(.hdate)
ZYQ(hdate) ;
  ;
  quit:+$get(hdate)'?5n ""   ; quit if argument passed in, isn't in the format
  set date=$zdate(hdate,1)   ; returned AS MM/DD/YY
  set month=+date ; convert date string to numeric, returns month alone.

  set quarter=$normalize((month+1)/3,0)
  quit "Q"_quarter
```

Note: From a best practice stand-point, functions like the example above are probably more intended for use by application code (rather than an administrator or developer working at the command-line prompt) and I would ideally prefer to implement the above as a method in a Utility-type class of an application.

Usage:

```
SAMPLES>WRITE $ZYQ($horolog)
“Q2”
```

Finally, you can define your own environment ’ s special variables. Special variables are variables maintained by the system.

You can retrieve, and in some cases, set, their value. A common read-only special variable is, for example, $ZVERSION which returns a string denoting the currently installed version. A special variable you can set is $NAMESPACE, which has the effect of switching namespaces for you automatically.

If you implement a special variable, that records a value, you are responsible for storing that value somewhere.

The following is an implementation of a special variable that returns the number of seconds elapsed since midnight

```
%ZLANGV00   ; Custom extended Special variables - SP
   ;
   Quit   ; quit nicely if this routine is actually invoked directly.

   ; Special variable representing seconds since midnight
ZHTIME() quit $$ZHT()
ZHT()   quit $piece($horolog,",",2)
```

Usage:

```
SAMPLES>Write !,"Good ",$select($zht>43200:"Afternoon.",1:"Morning.")
```

Conclusion and final thoughts:

Here are a couple of things to bare in mind if you intent to adopt this functionality.

I believe it is worth coming up with a handful of useful commands and socialise your proposal with your development team for approval and review, then finally, implementing as required. This is because once you implement %ZLANG* routines in your environment, they are available to all, and in all namespaces.

I do believe that a small set is warranted, don ’ t go overboard, and definitely only if there is no native alternative.

Keep commands short, with the long version (if you implement it) no more than 6-8 characters.

Language extensions are inherently slower calling native commands, functions or variables. Whilst you can invoke your new features in your application, as part of your application, it might be worth designating their use to command line debugging or administration, choosing instead to use the native commands within the application code.

When you use your extended command, functions or variables, they behave just like native ones, which means they are case insensitive, and, obey other structures like post conditional expressions. Eg write:X>1 $zht

Be careful not to define customer extensions that overlap existing native commands, functions or special variables, as, your version will not work.

Be careful when implementing your custom code, that it does not modify state that you would expect to remain the same, after your code runs - for example, special variables like $ZREFERENCE (the current global reference) or $TEST (truth value resulting from the last command using the timeout option) should not change if you execute your custom command.

Finally - the documentation on this topic can be found here:

http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=GSTUcustomize#GSTUcustomize_zlang

Steve

#Best Practices #Languages #Caché