InterSystems CACHÉ®

Using the Caché ^%ZSTART and ^%ZSTOP Routines

Version 2009.1
30 June 2009

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

Using the Caché ^%ZSTART and ^%ZSTOP Routines
Caché Version 2009.1 30 June 2009
Copyright © 2009 InterSystems Corporation
All rights reserved.

This book was assembled and formatted in Adobe Page Description Format (PDF) using tools and information from the following sources: Sun Microsystems, RenderX, Inc., Adobe Systems, and the World Wide Web Consortium at www.w3c.org. The primary document development tools were special-purpose XML-processing applications built by InterSystems using Caché and Java.

Table of Contents

Using the Caché ^%ZSTART and ^%ZSTOP Routines **1**

1 Introduction 1

2 Design Considerations 2

3 Enabling %ZSTART and %ZSTOP 3

4 Debugging ^%ZSTART and ^%ZSTOP 4

5 An Example Implementation 4

5.1 Routine: ^%ZSSUtil 5

5.2 Routine: ^%ZSTART 7

5.3 Routine: ^%ZSTOP 9

Using the Caché ^%ZSTART and ^%ZSTOP Routines

1 Introduction

Caché, like many systems, provides a mechanism that calls one or more user-written routines when certain noteworthy events happen. The processing for each event is usually treated as a subroutine of the system itself. In Caché, there are two such routines: `^%ZSTART` and `^%ZSTOP`. Each defines one or more labels from a set of four predefined by the system. These labels become entry points into each routine that are called when that class of events happens:

- SYSTEM: Caché as a system starts or stops
- LOGIN: A user performs a login or logout
- JOB: A **JOB** begins or ends
- CALLIN: An external program begins or completes a **CALLIN**

The appropriate label is called in `^%ZSTART` if the activity is beginning, and in `^%ZSTOP` if is it ending.

These entry points, though they function as subroutines of Caché, are not intended to do complex calculations or run for long periods of time. Long calculations or potentially long operations like network accesses will delay the completion of the activity until the called routine returns. In this case, users may take a long (elapsed) time to login, or JOB throughput may be curtailed because they take too long to start.

Note: These entrypoints are called as part of normal Caché operation. This means that an external event which terminates Caché abnormally, such as a power failure on the platform where Caché is executing, will not generate a call to %ZSTOP.

Note: When a system implements %ZSTOP, and an application implements one or more $HALT routines, the %ZSTOP code is not executed until the last $HALT terminates with a HALT command. The failure of a $HALT routine to issue its own HALT command can prevent the %ZSTOP code from running.

2 Design Considerations

Because `^%ZSTART` and `^%ZSTOP` run in a somewhat restricted environment, the designer must keep several things in mind, namely:

- The routines must be written in Caché ObjectScript.
- There are no values passed as arguments when any of the routine entry points are called. If different algorithms are applicable in various circumstances, the called entry point must determine what to do by examining data external to the routine: global, system variables, the return values from **$ZUTIL** routines, and so on.
- Take care to make certain the routines are well-behaved under all possible conditions. They should be written defensively. That is, they should check to make sure that all the resources needed to complete their task are at hand and, if possible, reserved to them before computation starts. Errors which occur are reported as failures of that system function so it is important to think about the design from the viewpoint of error containment and handling. Failure to properly account for recovery in the face of missing resources or the presence of errors has varied consequences: Caché may fail to start; major functions such as Studio may act strangely; or more subtle and insidious consequences may occur which are not immediately detected. It is strongly recommended that these routines be carefully written and debugged under simulated conditions, and then tested under simulated environment conditions before being put into production systems.
- No assumption should be made that conditions found during a previous invocation or a different entry point are still valid. Between successive calls to `JOB^%ZSTART`, for example, a file used by the prior call could have been deleted before this call occurred.
- Each entry point should perform its task efficiently. If part of the task is potentially long-running, enough information to complete it should be queued for later disposition by another part of your application.
- If an entry point wishes to have data around persistently for, say, statistical purposes, it must use something like a global or an external file to hold the data.
- The routines should make minimal assumptions about the environment they are running in. A developer of one of these routines cannot, for example, assume that the program will always be executed under a specific job number. The designer cannot assume that the various entry point will be called in a specific order. The sequence of bringing up the multiple processes that implement Caché is rarely deterministic.
- The routine cannot assume that it is being called at a specific point during the system startup. The sequence of events during startup may change from release to release, or even from restart to restart.
- With a few exceptions, the routine must leave things as it found them. As an illustration of this principle, reassigning the value of *$IO* in the subroutine without saving and restoring it upon entry

and exit is an almost certain source of error. The calling routine has no way of knowing that such things are changed, and it is very difficult for the caller to defend against any possible change to the execution environment. Therefore, the burden of not disturbing the system processing context lies on the subroutine being called.

The general exceptions to the no-changes rule are that changing process-local values specific to an application or installation are allowed. For example, the `SYSTEM^%ZSTART` entry point may invoke one or more of the **$ZUTIL(69)** subfunctions to set system-wide defaults. Similarly, for application testing, it could call **$ZUTIL(71,date)** to set the date to a specific value to validate end-of-month processing.

- `^%ZSTOP` cannot contain references to globals in remote databases. At the time it is called, some of these may no longer be accessible.
- If %Z* routines are mapped to a database different from CACHESYS, then Caché will attempt to execute `^%ZSTART` and `^%ZSTOP` from that database, not from CACHESYS. Caché will, of course, make certain that the calling routine has the appropriate access to that database beforehand. It is the responsibility of the administrator to ensure that the routine has access to any application globals and mappings that it requires from that namespace.
- `^%ZSTART` and `^%ZSTOP` are run with *$USERNAME* set to "%System" and *$ROLES* set to "%All" . If the administrator wishes to run the startup or shutdown routines with a different username, the routine should call

  ```
  $SYSTEM.Security.Login(<username>)
  ```

  to set the desired name. Any additional processes JOBbed by the startup routines will inherit the same username (and roles) as the initiating process.

Note: On upgrades, Caché preserves only the %Z* routines that are mapped to the CACHESYS database, and if the .INT or .MAC code is available, recompiles them. Preservation of routines in other databases are the responsibility of the site administrator.

3 Enabling %ZSTART and %ZSTOP

`^%ZSTART` and `^%ZSTOP` must reside (and therefore be compiled) in the `%SYS` namespace. Their mere presence is not sufficient, however, for Caché to automatically begin using them. Instead, individual entry points are enabled or disabled via the System Management Portal.

Once the routines have been designed, developed, compiled, and are ready to be tested, individual entry points may be enabled through the management portal. Navigate to the **[Home] > [Configuration] > [Startup Settings]** page and edit the appropriate individual settings:

- SysStart, SysHalt

- ProcessStart, ProcessHalt
- JobStart, JobHalt
- CallinStart, CallinHalt

As an illustration, suppose we wish to have something every time a background process starts. The appropriate entry point, `JOB^%ZSTART`, will be invoked by Caché when *ALL* the following are true:

- The desired routine entry point is enabled; that is, the value of the JobStart setting in the **[Home] > [Configuration] > [Startup Settings]** page is true
- `^%ZSTART` is compiled and loaded in the local Caché system pointed to by the management portal.
- The routine contains the **JOB** entry point.

4 Debugging `^%ZSTART` and `^%ZSTOP`

The opportunities for debugging `^%ZSTART` and `^%ZSTOP` in their final environment are very limited. If an error occurs, the error message that would be written to the TERMINAL in an interactive session is instead directed to the operator console log. This file is "cconsole.log" and is found in the Caché Manager's directory.

The message indicates the reason for failure and location where the error was detected. This may be different from the place where the error in the program logic or flow actually occurred. The developer is expected to deduce the nature and location of the error from the information provided, or modify the routine so that future tests provide more evidence as to the nature of the error.

To deactivate one or more of the entry points, use the same procedure but change the value to "false."

Remember that `^%ZSTART` and `^%ZSTOP` (as well as any supporting routines) are stored persistently. To remove all traces of them, delete them through the System Management Portal.

Note: It is strongly recommended that you disable the entry point options via the System Management Portal *BEFORE* deleting the routines. If the portal warns that a restart of Caché is needed for them to take effect, do this as well before proceeding. This guarantees that none of the entry points are being executed while they are being deleted.

5 An Example Implementation

The following example demonstrates a simple log for tracking system activity. It consists of a common utility for writing log entries called `^%ZSSUtil` as well as separate routines for `^%ZSTART` and

`^%ZSTOP`. Because the `^%ZSTOP` routine mirrors many of the activities of `^%ZSTART`, most of the discussion is about `^%ZSTART` and `^%ZSSUtil`. All three routines are shown, however.

Note: The code in these examples has been written to aid understanding, not to illustrate clever usages of ObjectScript or be "as tight as possible."

5.1 Routine: `^%ZSSUtil`

This routine has two public entry points. One is used to write a single line to the operator console log. The other is used to write a list of name-value pairs to a local log file. Both the operator console log and the local log file reside in the Caché Manager directory. The name of this directory is returned by the **ManagerDirectory** method of the %Library.File class.

```
%ZSSUtil ;
    ; this routine packages a set of subroutines
    ; used by the %ZSTART and %ZSTOP entry points
    ;
    ; does not do anything if invoked directly
    quit

#Define Empty ""
#Define OprLog 1

WriteConsole(LineText) PUBLIC ;
    ; write the line to the console log
    ; by default the file cconsole.log in the MGR directory
    new SaveIO

    ; save the current device and open the operator console
    ; set up error handling to cope with errors
    ; there is little to do if an error happens
    set SaveIO = $IO
    set $ZTRAP = "WriteConsoleExit"
    open $$$OprLog
    use $$$OprLog
    ; we do not need an "!" for line termination
    ; because each WRITE statement becomes its
    ; own console record (implicit end of line)
    write LineText
    ; restore the previous io device
    close $$$OprLog
    ; pick up here in case of an error
WriteConsoleExit ;
    set $ZTRAP = ""
    use SaveIO
    quit

WriteLog(rtnname, entryname, items) PUBLIC ;
    ; write entries into the log file
    ; the log is presumed to be open as
    ; the default output device
    ;
    ; rtnname: distinguishes between ZSTART & ZSTOP
    ; entryname: the name of the entry point we came from
    ; items: a $LIST of name-value pairs
    new ThisIO, ThisLog
    new i, DataString

    ; preserve the existing $IO device reference
    ; set up error handling to cope with errors
    ; there is little to do if an error happens
    set ThisIO = $IO
    set $ZTRAP = "WriteLogExit"

    ; construct the name of the file
    ; use the month and day as part of the name so that
    ; it will create a separate log file each day
    set ThisLog = "ZSS"
                _ "-"
                _ $EXTRACT($ZDATE($HOROLOG, 3), 6, 10)
                _".log"

    ; and change $IO to point to our file
    open ThisLog:"AWS":0
    use ThisLog

    ; now loop over the items writing one line per item pair
    for i = 1 : 2 : $LISTLENGTH(items)
    {
        set DataString = $LISTGET(items, i, "*MISSING*")
        if ($LISTGET(items, (i + 1), $$$Empty) '= $$$Empty)
        {
            set DataString = DataString
                           _ ": "
                           _ $LISTGET(items, (i + 1))
        }
        write $ZDATETIME($HOROLOG, 3, 1),
              ?21, rtnname,
              ?28, entryname,
              ?35, DataString, !
    }

    ; stop using the log file and switch $IO back
    ; to the value saved on entry
    close $IO
    ; pick up here in case of an error
WriteLogExit ;
    set $ZTRAP = ""
    use ThisIO
    quit
```

5.1.1 Explanation

Label: `^%ZSSUtil`

This routine (as well as the others) begins with a QUIT command so that it is benign if invoked via

```
    do ^%ZSSUtil
```

The #DEFINE sequence cosmetically provides named constants in the body of the program. In this instance, it names the empty string and the device number of the operator console log.

Label: `WriteConsole^%ZSSUtil`

The entry point is very simple. It is designed for low volume output, and as a minimally intrusive routine to use for debugging output.

It takes a single string as its argument and writes it to the operator console log. However, it must take care to preserve and restore the current *$IO* attachment across its call.

One aspect of the console device is that each item sent to the device results in a separate record being written to the console log. Thus,

```
    WRITE 1, 2, 3, !
```

results in four records being written. The first three consist of a single digit and the fourth is a blank line. If multiple items are desired on a line, it is the responsibility of the caller to concatenate them into a string.

Label: `WriteLog^%ZSSUtil`

This is the more complex of the two routines. It can be called by any entry point within `^%ZSTART` or `^%ZSTOP`. The information needed to report on whose behalf it is operating is supplied by the first two arguments. The third argument is a **$LIST** of name-value pairs to be written to the log.

This entry point starts by constructing the name of the file it will use. To make log management easier, the name contains the month and day when the routine is entered. Therefore, calls to WriteLog will create a new file whenever the local time crosses midnight. because the name is determined only at the time of the call, all the name-value pairs passed as the argument will be displayed in the same file.

Once the name has been constructed, the current value of *$IO* is saved for later use and the output device is switched to the named log file. The parameters used for the **OPEN** command ensure that the file will be created if it is not there. The timeout of zero indicates that Caché will try a single time to open the file and fail if it cannot.

Once the file has been opened, the code loops over the name value pairs. For each pair, the caller routine name and the entry point name are written followed in the line by the name-value pair. (If the value part is the empty string, only the name is written.) Each pair occupies one line in the log file. The first three values on each line are aligned so they appear in columns for easier scanning.

When all the pairs have been written, the log file is closed, the previous value *$IO* is restored and control returns to the caller.

5.2 Routine: `^%ZSTART`

This routine contains the entry point actually invoked by Caché. It uses the services of `^%ZSSUtil` just described. All the entry points act more or less the same, they place some information in the log. The SYSTEM entry point has been made slightly more elaborate than the others. It places information in the Operator console log as well.

```
%ZSTART ; User startup routine.

#Define ME "ZSTART"
#Define BgnSet "Start"
#Define Empty ""

    ; cannot be invoked directly
    quit

SYSTEM ;
    ; Cache starting
    new EntryPoint, Items

     set EntryPoint = "SYSTEM"

     ; record the fact we got started in the console log
     do WriteConsole^%ZSSUtil((EntryPoint
                          _ "^%"
                          _ $$$ME
                          _ " called @ "
                          _ $ZDATETIME($HOROLOG, 3)))

    ; log the data accumulate results
     set Items = $LISTBUILD($$$BgnSet, $ZDATETIME($HOROLOG, 3),
                      "Job", $JOB,
                      "Computer", $ZUTIL(110),
                      "Version", $ZVERSION,
                      "StdIO", $PRINCIPAL,
                      "Namespace", $ZUTIL(5),
                      "CurDirPath", $ZUTIL(12),
                      "CurNSPath", $ZUTIL(12, ""),
                      "CurDevName", $ZUTIL(67, 7, $JOB),
                      "JobType", $ZUTIL(67, 10, $JOB),
                      "JobStatus", $ZHEX($ZJOB),
                      "StackFrames", $STACK,
                      "AvailStorage", $STORAGE,
                      "UserName", $ZUTIL(67, 11, $JOB))
    do WriteLog^%ZSSUtil($$$ME, EntryPoint, Items)

    quit

LOGIN ;
    ; a user logs into Cache (user account or telnet)
    new EntryPoint, Items

    set EntryPoint = "LOGIN"
     set Items = $LISTBUILD($$$BgnSet, $ZDATETIME($HOROLOG, 3))
    do WriteLog^%ZSSUtil($$$ME, EntryPoint, Items)
    quit

JOB ;
    ; JOB'd process begins
    new EntryPoint, Items

     set EntryPoint = "JOB"
     set Items = $LISTBUILD($$$BgnSet, $ZDATETIME($HOROLOG, 3))
    do WriteLog^%ZSSUtil($$$ME, EntryPoint, Items)
    quit

CALLIN ;
    ; a process enters via CALLIN interface
    new EntryPoint, Items

     set EntryPoint = "CALLIN"
     set Items = $LISTBUILD($$$BgnSet, $ZDATETIME($HOROLOG, 3))
    do WriteLog^%ZSSUtil($$$ME, EntryPoint, Items)
    quit
```

5.2.1 Explanation

Label: `^%ZSTART`

As noted in the discussion of `^%ZSSUtil`, this routine begins with a **QUIT** command so that it is benign if invoked as a routine rather than beginning its execution properly at one of its entry points.

This routine also defines named constants (as macros) for its own name, a starting string and the empty string.

Label: SYSTEM^%ZSTART

because `^%ZSSUtil` takes care of most of the detail of writing to the log file(s), this routine consists mostly of collecting information and then invoking the proper entry point in `^%ZSSUtil`. It constructs a string giving its routine name, entry point, and the date and time it was invoked. Then it calls `WriteConsole^%ZSSUtil` to place it in the operator console log.

Afterward, it constructs a list of name-value pairs that it wishes to be displayed. It passes this to `WriteLog^%ZSSUtil` to place into the local log file. Then it returns to its caller.

Labels: LOGIN^%ZSTART, JOB^%ZSTART, CALLIN^%ZSTART

Unlike the SYSTEM entry point, these do not place any information in the operator console log. They construct a short list of items, enough to identify that they were invoked, and then use `WriteLog^%ZSSUtil` to record it.

5.3 Routine: ^%ZSTOP

This routine contains the entry point actually invoked by Caché. Like `^%ZSTART`, it uses the services of `^%ZSSUtil`. The entry points have the same names as those in `%ZSTART` and serve similar functions. As with `^%ZSTART`, only the SYSTEM entry point writes to the operator console log. All routines note their activity in the local log.

```
%ZSTOP ; User shutdown routine.

#Define ME "ZSTOP"
#Define EndSet "End"
#Define Empty ""

    ; cannot be invoked directly
    quit

SYSTEM ; Cache stopping
    new EntryPoint

    set EntryPoint = "SYSTEM"
     ; record termination in the console log
     do WriteConsole^%ZSSUtil((EntryPoint
                               _ "^%"
                               _ $$$ME
                               _ " called @ "
                               _ $ZDATETIME($HOROLOG, 3)))
     ; write the standard log information
    do Logit(EntryPoint, $$$ME)
    quit

LOGIN ; a user logs out of Cache (user account or telnet)
    new EntryPoint

    set EntryPoint = "LOGIN"
    do Logit(EntryPoint, $$$ME)
    quit

JOB ; JOB'd process exits.
    new EntryPoint

    set EntryPoint = "JOB"
    do Logit(EntryPoint, $$$ME)
    quit
```

```
CALLIN ; process exits via CALLIN interface.
    new EntryPoint

    set EntryPoint = "CALLIN"
    do Logit(EntryPoint, $$$ME)
    quit

Logit(entrypoint, caller) PRIVATE ;
    ; common logging for exits

    new items

     set items = $LISTBUILD($$$EndSet, $ZDATETIME($HOROLOG, 3))
    do WriteLog^%ZSSUtil(caller, entrypoint, items)
    quit
```

5.3.1 Explanation

This routine mimics the functions of `^%ZSTART`. Refer to the description, example, and explanation of `^%ZSTART` for information.