



Using the Caché ^%R Routine

Version 5.2
01 September 2006

Using the Caché ^%R Routine

Caché Version 5.2 01 September 2006

Copyright © 2006 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.



The Caché product and its logos are registered trademarks of InterSystems Corporation.



The Ensemble product and its logos are registered trademarks of InterSystems Corporation.



The InterSystems name and logo are trademarks of InterSystems Corporation.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

Caché, InterSystems Caché, Caché SQL, Caché ObjectScript, Caché Object, Ensemble, InterSystems Ensemble, Ensemble Object, and Ensemble Production are trademarks of InterSystems Corporation. All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Customer Support

Tel: +1 617 621-0700

Fax: +1 617 374-9391

Email: support@InterSystems.com

Table of Contents

Using the Caché ^%R Routine	1
ParseRoutineName^%R	5
ROUTINE^%R	7
FMterr^%R	11
CHECK^%R	12
DATE^%R	14
DEL^%R	16
EXISTS^%R	17
LANG^%R	18
LANGSET^%R	19
LENGTH^%R	20
SIZE^%R	22
LINE^%R	23
LINESET^%R	25
LOCK^%R	29
UNLOCK^%R	29
VERMAX^%R	30
VERMAXSET^%R	33
VERSION1^%R	34
VERSION^%R	35

List of Tables

Name Interpretation	4
General Routine Manipulation	4
Version Control	5

Using the Caché ^%R Routine

Background

In earlier releases of Caché, the generation and manipulation of routines by other routines was done incrementally. That is, the controlling routine would use [ZLOAD](#), [ZINSERT](#), [ZREMOVE](#), and related commands to build up, manipulate, compile and save routines on a line-by-line basis.

While this approach provided excellent error isolation, it proved inefficient. As the ObjectScript compiler evolved, the relative cost associated with the initialization and termination of a sophisticated compiler became much larger than the cost associated with compiling one line of code. Consequently, the **%R** routine was introduced in Caché Version 4 to improve the compilation speed of programmatically generated routines. The **%R** routine made it possible to treat the generated routines as units rather than dealing with them line-by-line.

Although it is still possible to manipulate routines one line at a time, **%R** is the preferred path.

Note: The two approaches are compatible with one caveat. If an application uses **\$ZUTIL(62, 0)** to determine whether there was a syntax error after each call to **ZINSERT**, this will always report that no errors are present. Previously, **ZINSERT** caused compilation to occur upon each insert. Now, compilation is delayed until the **ZSAVE** command is executed. However, **ZSAVE** does not report errors via the **\$ZUTIL(62, 0)** mechanism.

Routine Names

The use of a string to supply the name of a routine occurs in many of the entry points of **%R**. The general approach to resolving this name into its components parts is discussed here.

In general terms, the routine name consists of three parts, each separated from the others by a period: the base name, the routine extension, and the version of the source specified by the preceding two elements. However, uncertainties in the interpretation of names occur because of the following:

- a namespace may be included as part of the routine name
- each of the extension and version components is optional and may not be supplied in a given invocation

-
- under certain circumstances, the wildcard character (*) is permitted as all or part of each component
 - as of Caché Version 4, periods are valid as part of the base name (they specify the package which contains the routine).

To achieve backward compatibility for legacy programs evolving to use the ^%R functionality, the following approach is used to resolve the string into the component parts of base name, extension, version, and namespace:

1. The namespace component is initialized to the empty string.
2. If either of the characters "]" or "]" occurs in the string, the format is assumed to be of the form
 - ["<NAMESPACE>"]<REMAINDER>or
 - |"<NAMESPACE>"|<REMAINDER>

The namespace component is extracted from between the ("). The string is set to <REMAINDER> and processing continues.

3. If the string consists only of a wildcard character (*), then the extension and version components are set to the empty string. The base name is set to the wildcard character. Processing of the name is finished.
4. If the string is of the form "<TEXT>.*.*", then set the extension and version to "*", and the base name to <TEXT>. Processing is complete.
5. If the string is of the form "<TEXT>.*", then set the version to 0, the extension to "*", and the base name to <TEXT>. Processing is complete.
6. If the string is of the form "<TEXT>.<EXT>.*", and <EXT> is a valid extension, that is, one of
 - MAC
 - INT
 - INC
 - OBJ
 - BAS
 - COS

(case is ignored for this comparison), then set the version to "*", the extension to *<EXT>*, and the base name to *<TEXT>*.

Otherwise, set the version to 0, the extension to "*", and the base name to *<TEXT>*.*<EXT>*. Processing is complete.

7. If the string is of the form "*<TEXT>*.*<EXT>*.*<VER>*", and *EXT* is one of the valid extensions listed above, and *VER* is an integer (signed or unsigned), then set the version to *VER*, the extension to *<EXT>*, and the base name to *<TEXT>*.

Otherwise, set the version to 0, the extension to "*", and the base name to *<TEXT>*.*<EXT>*.*<VER>*. Processing is complete.

8. Otherwise, set the version to 0, the extension to "*", and the base name to the remaining string of characters. Processing is complete.

The preceding is only a general description of the algorithm for parsing names. For examples of how names are parsed, see the [ParseRoutineName](#) entry point below. This is the name parser used by the other entry points of **^%R**.

Language Encoding

In several entry points, an integer value is used to identify the language that the subject routine is written in. The encoding used for specifying the language is:

- 0 - Caché ObjectScript (default)
- 1 - DSM-11
- 2 - DTM
- 3 - Unused - Obsolete
- 4 - Unused - Obsolete
- 5 - DSM-VMS
- 6 - DSM-J
- 7 - DTM-J
- 8 - MSM
- 9 - Caché Basic

The values 1 through 8 inclusive are used to customize the compiler for subtle differences in the interpretation of the source code among these legacy systems.

Note: Current users of Caché are strongly advised to use either 0 or 9 as appropriate. The remaining values should only be used for maintaining or updating legacy applications.

Versioning

There are entry points in **%R** that provide a limited form of version control when used with the "B" option to the **ROUTINE** entry point. They provide for keeping a specified number of previous “ancestor” versions of the specified routine available, and for being able to recover them as needed. For details, see **VERMAX**, **VERMAXSET**, **VERSION1**, and **VERSION**.

Note: If you do not wish to use versioning at all, you may set the maximum to zero. Alternatively, you may kill the system globals **^rBACKUP**, as well as **^rMACSAVE** and **^rINCSAVE**.

Entry Points

The publicly supported entry points in **^%R** grouped functionally are:

Name Interpretation

Entry Point	Purpose
ParseRoutineName	Separate a routine name into its component parts

General Routine Manipulation

Entry Point	Purpose
ROUTINE	Compile, save, or load source code in database
FMTERR	Display errors
CHECK	Check syntax of source code
DATE	Get the saved routine date
DEL	Remove a routine from the database
EXISTS	Check to see if a routine exists
LANG	Get the source language indicator for a routine
LANGSET	Set the source language indicator for a routine
LENGTH	Count the source lines in a routine
SIZE	Count the source characters in a routine
LINE	Return a line from a saved routine

Entry Point	Purpose
LINESET	Insert/replace a new source line in a saved routine
LOCK	Attempt to gain exclusive use of a saved routine
UNLOCK	Gives up exclusive use of a saved routine

Version Control

Entry Point	Purpose
VERMAX	Returns the maximum number of versions to keep as backups
VERMAXSET	Sets the maximum number of versions to keep as backups
VERSION1	Returns the version number of the oldest backup
VERSION	Returns the version number corresponding to a relative version number

For more information on Caché ObjectScript functions generally, see the section “[Functions](#)” in [Using Caché ObjectScript](#).

ParseRoutineName^%R

```
ParseRoutineName^%R (rtn, .extent, .version, .namesp)
```

Parameters

rtn	The name of the target routine.
extent	A string holding the parsed extension.
version	A string giving the version number.
namesp	A string giving the namespace, if any, contained in the routine name.

Description

This function provides access to the routine name parser used by the other entry points of **^%R**.

Parameters

rtn

A string that specifies the name of the routine to be fetched from the database or stored into the database. A full or partial name of the routine may be given. The name is case-sensitive; the extension is not.

extent

This is an output argument and will hold the parsed extension.

version

This is an output argument and will hold the parsed version number.

namesp

If *rtn* contains an explicit namespace component, this output argument will contain it. Otherwise, it will be set to the empty string.

Remarks

This function is really an internal routine used by the other entry points of **^%R**. It is a PUBLIC entry point, however, and is described here to aid developers who wish to determine exactly how the parsing rules outlined in the introduction will be applied.

In addition to the output arguments, this function returns the base routine name as its result.

Examples

The following example demonstrates the use of **ParseRoutineName^%R** on a variety of routine names:

```

; Build a list of names
Set samples = $LISTBUILD("foo",
                        "foo.bar",
                        "foo.mac",
                        "foo*.bar",
                        "foo*.*.13",
                        "foo.mac.-234",
                        "^|" "DeltaQuadrant" |Voyager.int.1",
                        "[ ""^AlphaQuadrant" ]NCC.1701.MAC.4",
                        ^["twilight", "zone" ]Somewhere.INT.19")

; show the results
For i = 1 : 1 : $LISTLENGTH(samples)
{
    Set (Ext, Ver, Nsp) = "???"
    Set Input = $LIST(samples, i)
    Set BaseName = $$ParseRoutineName^%R(Input, .Ext, .Ver, .Nsp)
    Write Input, !
    Write ?3, "Base:", ?15, BaseName, !
    Write ?3, "Extension:", ?15, Ext, !
    Write ?3, "Version:", ?15, Ver, !
    Write ?3, "Namespace:", ?15, Nsp, !, !
}

```

ROUTINE^%R

```
ROUTINE^%R (rtn, .code, .errs, options, langmode, filedate, namesp,
iunlock)
```

Parameters

rtn	The name of the target routine.
code	A reference to an array of source code.
errs	A reference to a list of the errors detected during processing.
options	A string containing the processing options desired.
langmode	An indicator specifying the language of the source.
filedate	The timestamp to be used for the resulting file.
namesp	The namespace the routine is to be processed in.
iunlock	A switch indicating to immediately unlock the routine after processing.

Description

This function permits a Caché ObjectScript program to manipulate source code programmatically. Depending on the parameters passed to the function, an executing program may:

- Save source code for later use

- Retrieve saved source code
- Compile source code and optionally save the compiled object code

Parameters

rtn

An expression which evaluates to a string. The content of the string specifies the name of the routine to be fetched from the database or stored into the database. The full name of the routine must be given, that is, the name and extension separated by a period as in "SomeRoutine.MAC". The name is case-sensitive; the extension is not.

code

An array of the source code. This array supplies the source code of the routine when a routine is being compiled and/or saved. It receives the source when a routine is fetched.

The format of the array is as follows:

- *code*(0) contains the number of lines of source code involved in the operation.
- *code*(1) through *code*(*N*) contains individual lines of source code, where *N* is the value contained in *Code*(0).

errs

A list of the errors detected while attempting to carry out the processing specified by *options*. The argument, *err* is returned as a **\$LIST** of values. The values can be displayed by calling **FMTErr^%R** function.

options

A string consisting of a series of characters indicating the operations to be attempted on the routine. The permitted values and their meanings are:

- L - Load the previously saved routine into the *code* parameter.
- C - Compile the routine contained in *code*.
- D- Delete the saved source for the target routine.
- S - Save the routine source, and if there is a compiled version, save it as well.
- B - When saving routines of type "MAC" or "INT", create a backup version before doing the save.

The string is case-insensitive. The order in which operations are attempted is the order in which they occur in the string.

langmode

An integer indicating the language that the statements of *code* are written in.

filedate

The timestamp to be used as the modification date of the routine in Caché. The value of *filedate* is given in [\\$HOROLOG](#) format (see the *Caché ObjectScript Reference* for details). If *filedate* is not specified, the current date and time is used.

namesp

A string whose value is the namespace in which the specified operations are to take place. The parameter is optional. If unspecified, the current namespace is assumed.

iunlock

A boolean value where true (1) indicates that all locks for the routine are to be released as soon as processing is completed. False (the default) means to treat any locks held as in prior versions of Caché'.

Remarks

This routine attempts to carry out the specified operation(s) on the routine named in its first argument. It returns as its result a string of the form:

- $N^{Status1,Status2,...}$

The value of N is an indicator of the overall success or failure of the attempt. If N is 1, all operations completed successfully. If N is 0, one or more of the operations failed.

Status1, *Status2*, *Status3*, and so on represent short result summaries of the corresponding options.

Examples

The following example demonstrates the use of **ROUTINE^%R** to compile, save and execute a simple routine. It then executes the routine directly:

This example makes use of other entry points in **^%R** that are described elsewhere in this document.

WARNING! This example creates a routine in the SAMPLES namespace called AnExample.INT. If there is already a routine by that name in that namespace, it will be overridden.

```

; Change the namespace we will use
Znspace "SAMPLES"

; fill in the source code array
Set N = 0
Set code($INCREMENT(N)) = "AnExample ; An example routine"
; Note leading spaces on the following entries"
Set code($INCREMENT(N)) = "    Write "Starting AnExample", !"
Set code($INCREMENT(N)) = "    Set Y = 1"
Set code($INCREMENT(N)) = "    Set Z = 3"
Set code($INCREMENT(N)) = "    Write Y, "" + "", Z, "" = "", (Y + Z), !"
Set code($INCREMENT(N)) = "    Write "Finished AnExample", !"
Set code($INCREMENT(N)) = "    Quit"
Set code(0) = N

Set name = "AnExample"
Set ext = "INT"
Set routine = name _ "." _ ext
Set options = "CS"                ; Compile and Save
Set errors = ""                  ; empty list

; do it
Set return = $$ROUTINE^%R(routine, .code, .errors, options)

; show the simple result
Write "Compilation result: ", return, !
If (+return = "")
{
    ; format and display the errors
    Write $$FMTErr^%R(.errors, .code), !
}
Else
{
    ; run it
    Write "Calling AnExample", !
    Set entrypoint = name _ "^" _ name
    Do @entrypoint
    ; remove the source and object
    for suffix = "int", "OBJ"
    {
        Set component = name _ "." _ suffix
        Write "Removing ", component, ": ", $$DEL^%R(component), !
    }
}
}

```

FMTERR^%R

```
FMTERR^%R (.errs, .code, .lines)
```

Parameters

errs	A reference to a list of the errors detected during processing.
code	A reference to an array of source code.
lines	A reference to an array of text lines containing the interpretation of the errors found.

Description

This function converts the error data generated by an invocation of **ROUTINE^%R** or **CHECK^%R** into a form suitable for presentation to users.

Parameters

errs

A list of the errors returned by the invocation of **ROUTINE^%R** or **CHECK^%R**.

code

An array containing the source code. This array supplies the source code of the routine so that the displayed information may include the source line that caused the error as part of the text. Its format is the same as used by the **ROUTINE^%R** entry point. If it is not supplied, then the error message will only show the error location, but not the contents, of the line in error.

lines

A reference to an array that will be filled in with lines of text making up the error message(s). The format is the same as that of the source code array; *lines(0)* contains the number of error message lines in the remainder of the array.

If this parameter is not supplied, the resultant text will be returned as the value of the function invocation. Line breaks will be present in the returned string as occurrences of **\$CHAR(13, 10)**.

If *lines* is supplied, the individual array items represent separate lines. There are no internal line breaks. There is no result returned, that is, the routine should be invoked as a subroutine (via the [DO](#) command).

Remarks

This routine reformats the error list returned by **ROUTINE^%R** into more user-friendly form.

Examples

The following uses **CHECK^%R** on an illegal program to generate an error array. This array is then converted into text and displayed.

```
; fill in the source code array
Set N = 0
Set code($INCREMENT(N)) = "AnExample ; An example routine"
Set code($INCREMENT(N)) = "    Write "Starting AnExample", !"
Set code($INCREMENT(N)) = "    Write "A closing quote is missing here^, !"
Set code($INCREMENT(N)) = "    Quiet"
Set code(0) = N

Set errors = ""           ; empty list

; do it
Set return = $$CHECK^%R(.code, .errors)

; show the simple result
Write return, !
If (return = 0)
{
    ; format and display the errors
    Write !, "Errors as one text string", !
    Write $$FMTEERR^%R(.errors, .code), !
    Write !, "Errors as lines from an array", !
    Set lines = ""
    Do FMTEERR^%R(.errors, .code, .lines)
    for i = 1 : 1 : lines(0)
    {
        Write "errors(", i, ") = ", lines(i), !
    }
}
Else
{
    ; should never get here
    Write "Something is radically wrong", !
}
}
```

CHECK^%R

```
CHECK^%R (.code, .errs, langmode)
```

Parameters

code	A reference to an array of source code.
errs	A reference to a list of the errors detected during processing.
langmode	An indicator specifying the language of the source.

Description

This function performs a syntax check on the source code supplied and returns any errors found.

Parameters

code

An array containing the source code. This array supplies the source code of the routine to be checked for correct syntax. Its format is the same as used by the **ROUTINE^%R** entry point.

errs

A list of the errors reported by the syntax checker returned as a **\$LIST** of values. The values can be displayed by calling **FMTErr^%R** routine.

langmode

An integer which specifies the language that the statements of *code* are written in. The permitted values are the same as those permitted in the **ROUTINE^%R** routine.

Remarks

This routine is invoked as a function and returns either a 0 or 1 as its result. The interpretation of the result is:

- 1 - Success. There are no syntax errors present.
- 0 - Failure. Syntax errors were found and they are returned in the *errs* array.

Note: A return value of 1 only indicates that the supplied source will compile without error. There is no implication that the source code will execute without error, or produce the intended result.

Examples

See the example for the **FMTErr^%R** routine for an instance of **CHECK^%R**.

DATE^%R

```
DATE^%R (rtn, format, namesp)
```

Parameters

rtn	The name of the target routine.
format	The desired format of the time stamp.
namesp	The namespace the routine is saved in.

Description

This function returns the date and time that the named routine was saved in Caché.

Parameters

rtn

A string specifying the name of the routine to be fetched from the database or stored into the database. The full name of the routine must be given, that is, the name and extension separated by a period as in "SomeRoutine.MAC". The name is case-sensitive; the extension is not.

If the specified routine is not found, the function returns a null string.

format

An integer specifying the format the date and time are to be returned as. The allowed values are the same as for the [\\$ZDATETIME](#) function. The value given for *format* is used to format the *date* portion; the time is always shown as "HH:MM:SS".

In addition to the values permitted by [\\$ZDATETIME](#), a value of zero will return the time stamp in [\\$HOROLOG](#) format.

If the format value supplied is not one of those permitted, an error status will be returned as the value of the function. This is shown in the examples below.

namesp

A string whose value is the namespace in which the designated routine has been saved. The parameter is optional. If unspecified, the current namespace is assumed.

Remarks

This function and returns the time stamp associated with the routine at the time it was saved.

Note: This will not be the true date and time when the routine was saved if the caller of **ROUTINE^%R** provided a different value in the *filedate* parameter.

Examples

The following example displays the date and time that **%R** was saved in all the allowed formats.

```

; set up the data
Set name = "%R"
Set ext = "OBJ"
Set routine = name _ "." _ ext
Set namespace = "%CACHELIB"

; do it
Write "Timestamp for ", routine, !
for fmt = 0 : 1 : 12
{
    Set filedate = $$DATE^%R(routine, fmt, namespace)
    Write "Format ", fmt, ": ", ?10, filedate, !
}

```

This example shows the error status produced when the value of *format* is illegal.

```

; set up the data
Set name = "%R"
Set ext = "OBJ"
Set routine = name _ "." _ ext
Set namespace = "%CACHELIB"

; do it
Write "Timestamp for ", routine, !
Set filedate = $$DATE^%R(routine, 100, namespace)
Write "Value: ", ?10, filedate, !

```

Here is what happens when there is no routine by that name. A null string is returned as the value of the function.

```

; set up the data
Set name = "%NONEXISTENT"
Set ext = "OBJ"
Set routine = name _ "." _ ext
Set namespace = "%CACHELIB"

; do it
Write "Timestamp for ", routine, !
Set filedate = $$DATE^%R(routine, 1, namespace)
Write "Return length: ", $LENGTH(filedate), !

```

DEL^%R

```
DEL^%R (rtn, namesp)
```

Parameters

rtn	The name of the target routine.
namesp	The namespace the routine is saved in.

Description

This function removes the named routine from the Caché database.

Parameters

rtn

A string which gives the name of the routine to be fetched from the database or stored into the database. The name is case-sensitive; the extension is not.

Wildcards are in the name and the extension. For example, "Foo*.*" will remove all routines whose first three characters are "Foo" regardless of their extension.

namesp

A string whose value is the namespace in which the designated routine has been saved. The parameter is optional. If unspecified, the current namespace is assumed.

Remarks

This function remove the specified routines from the database. It returns a value of 1 if at least one routine was found and removed, and a value of 0 otherwise.

Examples

See the example for the [ROUTINE^%R](#) routine for an instance of **DEL^%R**.

EXISTS^%R

```
EXISTS^%R (rtn, namesp)
```

Parameters

rtn	The name of the target routine.
namesp	The namespace the routine is saved in.

Description

This function determines whether the specified routine exists within the Caché database.

Parameters

rtn

A string that specifies the name of the routine to be searched for.

Wildcards are allowed in the name and the extension.

namesp

A string whose value is the namespace in which the designated routine has been saved. The parameter is optional. If unspecified, the current namespace is assumed.

Remarks

This function returns 1 if there is a routine in the specified namespace which matches *rtn*. Otherwise, it returns 0.

Examples

Here we check to see if the subject of this article exists.

```

; set up the data
Set ext = "OBJ"
Set namespace = "%CACHELIB"

; do it
for basename = "%R", "ArbitraryName"
{
    Set routine = basename _ "." _ ext
    Set present = $$EXIST^%R(routine, namespace)
    Write routine, $SELECT(present:" exists", 1:" is missing"), "!", !
}

```

LANG[^]%R

```
LANG^%R (rtn, namesp)
```

Parameters

rtn	The name of the target routine.
namesp	The namespace the routine is saved in.

Description

This function returns the value of the language encoding for the specified routine.

Parameters

rtn

An expression which evaluates to a string. The content of the string specifies the name of the routine to be searched for.

namesp

A string whose value is the namespace in which the designated routine has been saved. The parameter is optional. If unspecified, the current namespace is assumed.

Remarks

This function returns the integer value which encodes the language mode of the specified routine. A list of the encodings is given at the beginning of this document.

If the routine extension is invalid, this function returns an empty string. If the extension is "MAC", the value returned is 0 (ObjectScript), regardless of whether the routine exists or not.

Examples

Here we check to see what some routines are written in.

```

; set up the data
Set namespace = "%CACHELIB"

set languages = $LISTBUILD("ObjectScript",
                           "DSM-11",
                           "DTM",
                           "",
                           "",
                           "DSM-VMS",
                           "DSM-J",
                           "DTM-J",
                           "MSM",
                           "Basic")

; do it
for routine = "%R.OBJ", "ArbitraryName.MAC", "SomeOther.RTN"
{
  Set code = $$LANG^%R(routine, namespace)
  If ($ISVALIDNUM(code))
  {
    Set lang = $LISTGET(languages, (code + 1), "")
    Set:(lang = "") lang = "Unknown"
    Write routine, " language: ", lang, !
  }
  Else
  {
    Write routine, ": invalid extension", !
  }
}

```

LANGSET^%R

LANGSET^%R (rtn, langmode)

Parameters

rtn	The name of the target routine.
langmode	An indicator specifying the new language of the source.

Description

This function returns the value of the language encoding for the specified routine.

Parameters

rtn

A string that specifies the name of the routine to be searched for.

langmode

An integer specifying the language that the statements of *rtn* are written in.

Remarks

This function changes the language encoding stored with the source of the named routine saved in the database. It is intended for those legacy applications that run on multiple platforms and need to adapt themselves to the environment at runtime.

This function returns 1 if the stored language mode was successfully changed, and 0 otherwise.

Examples

LENGTH^%R

```
LENGTH^%R (rtn, namesp)
```

Parameters

rtn	The name of the target routine.
namesp	The namespace the routine is saved in.

Description

This function returns the number of lines in the routine saved in the database.

Parameters

rtn

The name of the routine (as a string) to be searched for.

namesp

A string whose value is the namespace in which the designated routine has been saved. The parameter is optional. If unspecified, the current namespace is assumed.

Remarks

This function returns an integer value which is the number of lines in the named routine saved in the database. If the routine does not exist, the result of this function is 0.

Examples

The following creates a routine, and then saves it in the database. It erases its in-memory information on the routine, and queries the database for its info.

This example makes use of other entry points in ^%R that are described elsewhere in this document.

WARNING! This example creates a routine in the SAMPLES namespace called AnExample.INT. If there is already a routine by that name in that namespace, it will be overridden.

```

; Change the namespace we will use
Znspc "SAMPLES"

; fill in the source code array
Set N = 0
Set code($INCREMENT(N)) = "AnExample ; An example routine"
; Note leading spaces on the following entries"
Set code($INCREMENT(N)) = "    Write "Starting AnExample", !"
Set code($INCREMENT(N)) = ""
Set code($INCREMENT(N)) = "    Write "Some text", !"
Set code($INCREMENT(N)) = ""
Set code($INCREMENT(N)) = "    Write "Finished AnExample", !"
Set code($INCREMENT(N)) = "    Quit"
Set code(0) = N

Set routine = "AnExample.INT"
Set options = "S"                ; Save
Set errors = ""                  ; empty list

; do it
Set return = $$ROUTINE^%R(routine, .code, .errors, options)

; show the simple result
Write "Save result: ", return, !
If (+return = "")
{
    ; format and display the errors
    Write $$FMTErr^%R(.errors, .code), !
}
Else
{
    ; remove local info
    Kill code

    ; find out about it
    Write "Lines in ", routine, ": ", $$LENGTH^%R(routine), !

    ; remove the saved source
    Write "Removing ", routine, ": ", $$DEL^%R(routine), !
}

```

SIZE^%R

```
SIZE^%R (rtn, namesp)
```

Parameters

rtn	The name of the target routine.
namesp	The namespace the routine is saved in.

Description

This function returns the number of characters in the routine saved in the database.

Parameters

rtn

A string that gives the name of the routine to be searched for.

namesp

A string whose value is the namespace in which the designated routine has been saved. The parameter is optional. If unspecified, the current namespace is assumed.

Remarks

This function returns an integer value which is the number of characters in the named routine saved in the database. If the routine does not exist, the result of this function is 0.

Examples

The following creates a routine, and then saves it in the database. It erases its in-memory information on the routine, and queries the database for its info.

This example makes use of other entry points in ^%R that are described elsewhere in this document.

WARNING! This example creates a routine in the SAMPLES namespace called AnExample.INT. If there is already a routine by that name in that namespace, it will be overridden.

```

; Change the namespace we will use
Znspace "SAMPLES"

; fill in the source code array
Set N = 0
Set code($INCREMENT(N)) = "AnExample ; An example routine"
; Note leading spaces on the following entries"
Set code($INCREMENT(N)) = "    Write "AnExample", !"
Set code($INCREMENT(N)) = "    Quit"
Set code(0) = N

Set routine = "AnExample.INT"
Set options = "S"           ; Save
Set errors = ""           ; empty list

; do it
Set return = $$ROUTINE^%R(routine, .code, .errors, options)

; show the simple result
Write "Save result: ", return, !
If (+return = "")
{
    ; format and display the errors
    Write $$FMTERR^%R(.errors, .code), !
}
Else
{
    ; remove local info
    Kill code

    ; find out about it
    Write "Characters in ", routine, ": ", $$SIZE^%R(routine), !

    ; remove the saved source
    Write "Removing ", routine, ": ", $$DEL^%R(routine), !
}

```

LINE^%R

```
LINE^%R (rtn, linenum, namesp)
```

Parameters

rtn	The name of the target routine.
linenum	The number of the line desired.
namesp	The namespace the routine is saved in.

Description

This function returns a line of source from a routine saved in the database.

Parameters

rtn

The name of the routine to be searched for.

linenum

The number of the line in the routine whose contents is desired. The line number of the initial line of the routine is always 1.

namesp

A string whose value is the namespace in which the designated routine has been saved. The parameter is optional. If unspecified, the current namespace is assumed.

Remarks

This function returns the contents of the desired line as a string. If *linenum* does not satisfy the constraint that

- $1 \leq \textit{linenum} \leq \text{\$LENGTH}^{\text{\%R}}(\textit{rtn}, \textit{namesp})$

then the line does not exist and the function returns an empty string as its result. It is not possible to distinguish a non-existent line from a line containing no characters solely by using this function.

Examples

The following creates a routine, and then saves it in the database. It erases its in-memory information on the routine, and queries the database for its info.

This example makes use of other entry points in **^%R** that are described elsewhere in this document.

WARNING! This example creates a routine in the SAMPLES namespace called AnExample.INT. If there is already a routine by that name in that namespace, it will be overridden.

```

; Change the namespace we will use
Znspace "SAMPLES"

; fill in the source code array
Set N = 0
Set code($INCREMENT(N)) = "AnExample ; An example routine"
; Note leading spaces on the following entries"
Set code($INCREMENT(N)) = "    Write "AnExample", !"
Set code($INCREMENT(N)) = "    Quit"
Set code(0) = N

Set routine = "AnExample.INT"
Set options = "S"           ; Save
Set errors = ""           ; empty list

; do it
Set return = $$ROUTINE^%R(routine, .code, .errors, options)

; show the simple result
Write "Save result: ", return, !
If (+return = "")
{
    ; format and display the errors
    Write $$FMTERR^%R(.errors, .code), !
}
Else
{
    ; remove local info
    Kill code

    ; list the routine backwards
    For i = $$LENGTH^%R(routine) : -1 : 1
    {
        Write "Line ", i, ":: ", $$LINE^%R(routine, i), !
    }

    ; remove the saved source
    Write "Removing ", routine, ": ", $$DEL^%R(routine), !
}
}

```

LINESET^%R

```
LINESET^%R (rtn, linenum, linetext)
```

Parameters

rtn	The name of the target routine.
linenum	The number of the line to be replaced or added.
linetext	The contents of the new line.

Description

This function inserts the source text at the position indicated in the specified routine.

Parameters

rtn

A string which specifies the name of the routine to be modified.

linenum

The number of the line in the routine whose contents is desired. The line number of the initial line of the routine is 1.

linetext

A string of source text to be placed in the routine.

Remarks

This function inserts *linetext* in the routine at position *linenum*. If

- $linenum > \$\$LENGTH^{\%}R(rtn, namesp)$

then the routine source is effectively extended by appending

- $linenum - \$\$LENGTH^{\%}R(rtn namesp)$

empty lines to the source so that *linenum* is in the range

- $1 \leq linenum \leq \$\$LENGTH^{\%}R(rtn. namesp)$

Then the existing text at that line is replaced.

The function returns a result of 1 if the replacement was successful, and 0 otherwise.

Note: Replacing a line of source does not affect any corresponding object routine. To have the change take effect, the source routine must be recompiled.

Examples

The following creates a routine, and then compiles it and saves it in the database. It erases its in-memory information on the routine. It invokes the newly created routine.

Then it changes a line of the source and invokes the routine to demonstrate that no change has been made to the object.

Finally, it loads the new source from the database, re-compiles and saves it, and once more invokes the routine to show the effect of the change.

This example makes use of other entry points in ^%R that are described elsewhere in this document.

WARNING! This example creates a routine in the SAMPLES namespace called AnExample.INT. If there is already a routine by that name in that namespace, it will be overridden.

```

; Change the namespace we will use
Znspace "SAMPLES"

; fill in the source code array
Set N = 0
Set code($INCREMENT(N)) = "AnExample ; An example routine"
; Note leading spaces on the following entries"
Set code($INCREMENT(N)) = "    Write ""User: """"Hello, world""""", !"
Set code($INCREMENT(N)) = "    ; A dummy line to be replaced later"
Set ReplaceLoc = N
Set code($INCREMENT(N)) = "    Quit"
Set code(0) = N

Set name = "AnExample"
Set ext = "INT"
Set routine = name _ "." _ ext
Set options = "CS"           ; Compile and Save
Set errors = ""             ; empty list

; do it
Set return = $$ROUTINE^%R(routine, .code, .errors, options)
If (+return = "")
{
    ; format and display the errors
    Write $$FMTErr^%R(.errors, .code), !
    Quit
}

; run it
Set entrypoint = name _ "^" _ name
Do @entrypoint

Set NewLine = "    Write ""World: """"Hello yourself""""", !"
If ($$LINESET^%R(routine, ReplaceLoc, NewLine) '= 1)
{
    Write "Line replacement failed", !
    Quit
}

; show what is in the local array and in database
Write "Local: ", code(ReplaceLoc), !
Write "Saved: ", $$LINE^%R(routine, ReplaceLoc), !

; run it again to show same result
Do @entrypoint

; load, re-compile and save
Write "Updating...", !
Set return = $$ROUTINE^%R(routine, .code, .errors, "LCS")
If (+return = "")
{
    ; format and display the errors
    Write $$FMTErr^%R(.errors, .code), !
    Quit
}

; run it one last time to show change
Do @entrypoint

; remove the source and object
for suffix = "int", "OBJ"
{
    Set component = name _ "." _ suffix
    Write "Removing ", component, ": ", $$DEL^%R(component), !
}

```

LOCK^%R

```
LOCK^%R (rtn, timeout)
```

Parameters

rtn	The name of the target routine.
timeout	The number of seconds to wait before giving up the attempt.

Description

This function attempts to gain exclusive access to a routine saved in the database.

Parameters

rtn

The name of the routine to be locked as a string.

timeout

An integer giving the maximum number of seconds that can elapse before the routine will give up trying to acquire a lock on the routine. This parameter is optional; if it is not supplied the default timeout is 2.

Remarks

This attempts to gain exclusive access to the specified routine in the database. It returns 1 if the attempt was successful, and 0 if the lock could not be obtained.

Examples

UNLOCK^%R

```
UNLOCK^%R (rtn, iunlock)
```

Parameters

rtn	The name of the target routine.
iunlock	A switch indicating immediately unlock the routine after processing.

Description

This function undoes the action of **UNLOCK**.

Parameters

rtn

The name of the (presumably locked) routine as a string.

iunlock

A boolean value where true (1) indicates that all locks for the routine are to be released as soon as processing is completed. False (the default) means to treat any locks held as in prior versions of Caché'.

Remarks

This function gives up the lock to the routine it acquired via **UNLOCK**. It returns 1 if the attempt was successful, and 0 if the operation was unsuccessful as, for example, if the routine had not been previously locked.

Examples

VERMAX^%R

```
VERMAX^%R (extent, namesp)
```

Parameters

extent	The extent for which version information is desired.
namesp	The namespace this extent is saved in.

Description

This function returns the number of versions of routines with this extent that will be maintained.

Parameters

extent

A string giving the extent for which version information is desired.

namesp

The content of the string specifies the namespace to examine for the extent versioning information. If this is not supplied, the current namespace is assumed.

Remarks

This function examines the specified namespace for version information about the specified extent. If no version information has explicitly been set by an application before this call is made, then a system default value is used (currently 4).

If N is the value returned by the function, then there will be one current (most recent, or master) version and $N-1$ backup versions.

Note: The version numbers available as backups are not required to be a sequence of consecutive numbers. Intermediate versions may have been eliminated by calls to **DEL^%R**.

Examples

The following example illustrates the use of many of the entry points associated with version management.

WARNING! This example creates a routine in the SAMPLES namespace called AnExample.INT. If there is already a routine by that name in that namespace, it will be overridden.

```

; Change the namespace we will use
Znspace "SAMPLES"

; fill in the source code array
Set N = 0
Set code($INCREMENT(N)) = "AnExample ; An example routine"
Set code($INCREMENT(N)) = "    Write "Starting AnExample", !"
Set code($INCREMENT(N)) = "    Write "Line to be replaced", !"
Set lininx = N
Set code($INCREMENT(N)) = "    Write "Finished AnExample", !"
Set code($INCREMENT(N)) = "    Quit"
Set code(0) = N

Set name = "AnExample"
Set ext = "MAC"
Set routine = name _ "." _ ext
Set options = "BCS" ; Backup, Compile and Save
Set errors = "" ; empty list
Set abort = 0

; get the number of backups
Set maxvers = $$VERMAX^%R(ext)
Write "Default versions: ", maxvers, !
; increase it by one
Do VERMAXSET^%R(ext, (maxvers + 1))
; confirm it
set maxvers = $$VERMAX^%R(ext)
Write "Explicit versions: ", maxvers, !

; process the base version
Set return = $$ROUTINE^%R(routine, .code, .errors, options)
; show the simple result
Write "Compilation result: ", return, !
If (+return = "")
{
    ; format and display the errors
    Write $$FMterr^%R(.errors, .code), !
    set abort = 1
}
Else
{
    ; run it
    Write "Calling AnExample", !
    Set entrypoint = name _ "^" _ name
    Do @entrypoint
}
Quit:(abort)

; generate a bunch of versions
for i = 1 : 1 : 10
{
    ; process the new version
    Write !, "Loop ", i, !
    Set code(lininx) = "    Write "Iteration " _ i _ "", !"
    Set return = $$ROUTINE^%R(routine, .code, .errors, options)
    ; show the simple result
    Write "Compilation result: ", return, !
    If (+return = "")
    {
        ; format and display the errors
        Write $$FMterr^%R(.errors, .code), !
        set abort = 1
    }
    Else
    {
        ; run it
        Write "Calling AnExample", !
        Set entrypoint = name _ "^" _ name
    }
}

```

```

    Do @entrypoint
  }

  ; get the version boundaries
  Set oldest = $$VERSION1^%R(routine)
  Set vername = routine _ "." _ "-1"
  Set youngest = $$VERSION^%R(vername)
  Write "Oldest: ", oldest, "; ", "Youngest: ", youngest, !
}
; remove the source and object
for suffix = "int", "OBJ"
{
  Set component = name _ "." _ suffix
  Write "Removing ", component, ": ", $$DEL^%R(component), !
}
; reset the maximum versions
Kill ^rBACKUP(0,ext)
Quit

```

VERMAXSET^%R

VERMAX^%R (extent, max, namesp)

Parameters

extent	The extent for which version information is desired.
max	The maximum number of versions to be saved.
namesp	The namespace this extent is saved in.

Description

This function sets the number of versions of routines with this extent that will be maintained in the specified namespace.

Parameters

extent

The content of the string specifies the extent for which version information is desired.

max

An expression which evaluates to a positive, non-zero integer giving the number of versions to be maintained as backups.

namesp

The content of the string specifies the namespace to examine for the extent versioning information. If this is not supplied, the current namespace is assumed.

Remarks

This function sets the versioning information about the specified extent in the specified namespace. If *max* is set to 1, only the most recent version will be saved.

It returns 1 if the new maximum is set, and 0 otherwise.

Examples

See the example supplied with the **VERMAX^%R** entry point.

VERSION1^%R

```
VERSION1^%R (rtn, namesp)
```

Parameters

rtn	An expression which evaluates to a string giving the routine name.
namesp	The namespace this extent is saved in.

Description

This returns the number of the oldest version maintained as a backup.

Parameters

rtn

A string that supplies the name of the routine to whose oldest version the application is interested in. It includes the extent as part of the name. The name is case-sensitive; the extent is not. Any version number supplied as part of *rtn* is ignored.

namesp

An expression which evaluates to a string. The content of the string specifies the namespace to examine for the extent versioning information. If this is not supplied, the current namespace is assumed.

Remarks

This function examines the specified namespace looking for versions of the named routine which have been backed up. The version number of the oldest such routine is returned.

Examples

See the example supplied with the **VERMAX^%R** entry point.

VERSION^%R

```
VERSION^%R (rtn, namesp)
```

Parameters

rtn	An expression which evaluates to a string giving the routine name.
namesp	The namespace this extent is saved in.

Description

This returns the number of the version relative to the most recent version.

Parameters

rtn

A string which specifies the name of the routine the application is interested in. It includes the extent as part of the name. The name is case-sensitive; the extent is not. The version number is can be a signed value indicating its relationship to the most recent backup, for example, “SomeRoutine.MAC.-1” asks for the version number of the routine before the most recent one.

namesp

An expression which evaluates to a string. The content of the string specifies the namespace to examine for the extent versioning information. If this is not supplied, the current namespace is assumed.

Remarks

This returns the version number specified if the version part of the name is not relative. If the version portion of the name is relative, it returns the version number of the backup which is relative to the most recent version.

If the version specified does not exist, the routine returns 0.

Examples

See the example supplied with the **VERMAX^%R** entry point.