Using the Caché Callin API

Version 2018.1
2024-04-03

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

Using the Caché Callin API
Caché Version 2018.1 2024-04-03

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Tables

1

About This Book

See the Table of Contents for a detailed listing of the subjects covered in this document.

This book describes how to use the Caché Callin API, which offers an interface that you can use from within C or C++ programs to execute Caché commands and evaluate Caché expressions.

In order to use this book, you should be reasonably familiar with your operating system, and have significant experience with C, C++, or another language that can use the C/C++ calling standard for your operating system.

The following topics are covered:

- The chapter "The Callin Interface" describes the Callin interface, which you can use from within C programs to execute Caché commands and evaluate Caché expressions.
- The chapter "Using the Callin Functions" provides a quick summary of the Callin functions (with links to the full description of each function) catagorized according to the tasks they perform.
- The chapter "Callin Function Reference" contains detailed descriptions of all Caché Callin functions, arranged in alphabetical order.

Related Information

The Callin functions provide a very low-level programming interface. In many cases, you will be able to accomplish your objectives much more easily by using one of the standard Caché language bindings. For details, see the following sources:

- *Using C++ with Caché*
- *Using the Caché Managed Provider for .NET*
- *Using Java with Caché*

The Caché Callout Gateway is a programming interface that allows you to create a shared library with functions that can be invoked from Caché. Callout code is usually written in C or C++, but can be written in any language that supports C/C++ calling conventions.

- *Using the Caché Callout Gateway*

2

The Callin Interface

Caché offers a Callin interface you can use from within C programs to execute Caché commands and evaluate Caché expressions. This chapter describes this interface and includes the following sections:

- The callin.h Header File
- 8-bit and Unicode String Handling
- Using Caché Security Functions
- Using Callin with Multithreading
- Callin Programming Tips
- Running Sample Programs on Windows
- Running Sample Programs on UNIX® and Linux

The Callin interface permits a wide variety of applications. For example, you can use it to make ObjectScript available from an integrated menu or GUI. If you gather information from an external device, such as an Automatic Teller Machine or piece of laboratory equipment, the Callin interface lets you store this data in a Caché database. Although Caché currently supports only C and C++ programs, any language that uses the calling standard for that platform (UNIX®, Windows) can invoke the Callin functions.

See Using the Callin Functions for a quick review of Callin functions. For detailed reference material on each Callin function, see the Callin Function Reference.

2.1 The callin.h Header File

The callin.h header file defines prototypes for these functions, which allows your C compiler to test for valid parameter data types when you call these functions within your program. You can add this file to the list of #include statements in your C program:

```
#include "callin.h"
```

The callin.h file also contains definitions of parameter values you use in your calls, and includes various `#defines` that may be of use. These include operating-system–specific values, error codes, and values that determine how Caché behaves.

You can translate the distributed header file, callin.h. However, callin.h is subject to change and you must track any changes if you create a translated version of this file. InterSystems Worldwide Support Center does not handle calls about unsupported languages.

Return values and error codes

Most Callin functions return values of type int, where the return value does not exceed the capacity of a 16-bit integer. Returned values can be CACHE_SUCCESS, a Caché error, or a Callin interface error.

There are two types of errors:

- Caché errors — The return value of a Caché error is a positive integer.
- Interface errors — The return value of an interface error is 0 or a negative integer.

callin.h defines symbols for all Caché and interface errors, including CACHE_SUCCESS (0) and CACHE_FAILURE (-1). You can translate Caché errors (positive integers) by making a call to the Callin function **CacheErrxlate**.

2.2 8-bit and Unicode String Handling

Caché Callin functions that operate on strings have both 8-bit and Unicode versions. These functions use a suffix character to indicate the type of string that they handle:

- Names with an "A" suffix or no suffix at all (for example,**CacheEvalA** or **CachePopStr**) are versions that operate on local 8-bit encoded character strings.
- Names with a "W" suffix (for example,**CacheEvalW** or **CachePopStrW**) are versions for Unicode character strings on platforms that use 2–byte Unicode characters.
- Names with an "H" suffix (for example,**CacheEvalH** or **CachePopStrH**) are versions for Unicode character strings on platforms that use 4–byte Unicode characters.

For best performance, use the kind of string native to your installed version of Caché.

2.2.1 8-bit String Data Types

Caché supports the following data types that use local 8-bit string encoding:

- CACHE_ASTR — counted string of 8-bit characters
- CACHE_ASTRP — Pointer to an 8-bit counted string

The type definition for these is:

```
#define CACHE_MAXSTRLEN 32767
typedef struct {
   unsigned short  len;
   Callin_char_t   str[CACHE_MAXSTRLEN];
} CACHE_ASTR, *CACHE_ASTRP;
```

The CACHE_ASTR and CACHE_ASTRP structures contain two elements:

- len — An integer. When used as input, this element specifies the actual length of the string whose value is supplied in the str element. When used as output, this element specifies the maximum allowable length for the str element; upon return, this is replaced by the actual length of str.
- str — A input or output string.

CACHE_MAXSTRLEN is the maximum length of a string that is accepted or returned. A parameter string need not be of length CACHE_MAXSTRLEN nor does that much space have to be allocated in the program.

2.2.2 2–byte Unicode Data Types

Caché supports the following Unicode-related data types on platforms that use 2–byte Unicode characters:

- CACHEWSTR — Unicode counted string
- CACHEWSTRP — Pointer to Unicode counted string

The type definition for these is:

```
typedef struct {
   unsigned short len;
   unsigned short str[CACHE_MAXSTRLEN];
} CACHEWSTR, *CACHEWSTRP;
```

The CACHEWSTR and CACHEWSTRP structures contain two elements:

- len — An integer. When used as input, this element specifies the actual length of the string whose value is supplied in the str element. When used as output, this element specifies the maximum allowable length for the str element; upon return, this is replaced by the actual length of str.
- str — A input or output string.

CACHE_MAXSTRLEN is the maximum length of a string that is accepted or returned. A parameter string need not be of length CACHE_MAXSTRLEN nor does that much space have to be allocated in the program.

On Unicode-enabled versions of Caché, there is also the data type CACHE_WSTRING, which represents the native string type on 2–byte platforms. **CacheType** returns this type. Also, **CacheConvert** can specify CACHE_WSTRING as the data type for the return value; if this type is requested, the result is passed back as a counted Unicode string in a CACHEWSTR buffer.

2.2.3 4–byte Unicode Data Types

Caché supports the following Unicode-related data types on platforms that use 4–byte Unicode characters:

- CACHEHSTR — Extended Unicode counted string
- CACHEHSTRP — Pointer to Extended Unicode counted string

The type definition for these is:

```
typedef struct {
   unsigned int len;
   wchar_t str[CACHE_MAXSTRLEN];
} CACHEHSTR, *CACHEHSTRP;
```

The CACHEHSTR and CACHEHSTRP structures contain two elements:

- len — An integer. When used as input, this element specifies the actual length of the string whose value is supplied in the str element. When used as output, this element specifies the maximum allowable length for the str element; upon return, this is replaced by the actual length of str.
- str — A input or output string.

CACHE_MAXSTRLEN is the maximum length of a string that is accepted or returned. A parameter string need not be of length CACHE_MAXSTRLEN nor does that much space have to be allocated in the program.

On Unicode-enabled versions of Caché, there is also the data type CACHE_HSTRING, which represents the native string type on 4–byte platforms. **CacheType** returns this type. Also, **CacheConvert** can specify CACHE_HSTRING as the data

type for the return value; if this type is requested, the result is passed back as a counted Unicode string in a CACHEHSTR buffer.

2.2.4 System-neutral Symbol Definitions

The allowed inputs and outputs of some functions vary depending on whether they are running on an 8-bit system or a Unicode system. For many of the "A" (ASCII) functions, the arguments are defined as accepting a CACHESTR, CACHE_STR, CACHESTRP, or CACHE_STRP type. These symbol definitions (without the "A", "W", or "H") can conditionally be associated with either the 8-bit or Unicode names, depending on whether the symbols CACHE_UNICODE and CACHE_WCHART are defined at compile time. This way, you can write source code with neutral symbols that works with either local 8-bit or Unicode encodings.

The following excerpt from callin.h illustrates the concept:

```
#if defined(CACHE_UNICODE) /* Unicode character strings */
#define   CACHESTR      CACHEWSTR
#define   CACHE_STR     CACHEWSTR
#define   CACHESTRP     CACHEWSTRP
#define   CACHE_STRP    CACHEWSTRP
#define   CACHE_STRING  CACHE_WSTRING

#elif defined(CACHE_WCHART)  /* wchar_t character strings */
#define   CACHESTR      CACHEHSTR
#define   CACHE_STR     CACHEHSTR
#define   CACHESTRP     CACHEHSTRP
#define   CACHE_STRP    CACHEHSTRP
#define   CACHE_STRING  CACHE_HSTRING

#else                  /* 8-bit character strings */
#define   CACHESTR      CACHE_ASTR
#define   CACHE_STR     CACHE_ASTR
#define   CACHESTRP     CACHE_ASTRP
#define   CACHE_STRP    CACHE_ASTRP
#define   CACHE_STRING  CACHE_ASTRING
#endif
```

2.3 Using Caché Security Functions

Two functions are provided for working with Caché passwords:

- **CacheSecureStart** — Similar to **CacheStart**, but with additional parameters for password authentication. The **CacheStart** function is now deprecated. If used, it will behave as if **CacheSecureStart** has been called with NULL for Username, Password, and ExeName. You cannot use **CacheStart** if you need to use some form of password authentication.
- **CacheChangePassword** — This function will change the user's password if they are using Caché authentication (it is not valid for LDAP/DELEGATED/Kerberos etc.). It must be called before a Callin session is initialized.

There are **CacheSecureStart** and **CacheChangePassword** functions for ASCII "A", Unicode "W", and Unicode "H" installs. The new functions either narrow, widen or "use as is" the passed in parameters, store them in the new Callin data area, then eventually call the **CacheStart** entry point.

CacheStart and **CacheSecureStart** *pin* and *pout* parameters can be passed as NULL, which indicates that the platform's default input and output device should be used.

2.4 Using Callin with Multithreading

Caché has been enhanced so that Callin can be used by threaded programs running under some versions of Windows and UNIX® (see "Other Supported Features" in the online *InterSystems Supported Platforms* document for this release for a list). A program can spawn multiple threads (pthreads in a UNIX® environment) and each thread can establish a separate connection to Caché by calling **CacheSecureStart**. Threads may not share a single connection to Caché; each thread which wants to use Cache must call **CacheSecureStart**. If a thread attempts to use a Callin function and it has not called **CacheSecureStart**, a `CACHE_NOCON` error is returned.

A threaded application must link against cachet.o or the shared library, cachet.so. On UNIX® and Linux they may alternatively load the shared library dynamically. On Windows, due to the implementation of thread local storage the cachet.dll library cannot be dynamically loaded. The program should be careful not to exit until all of the threads which have entered Caché have called **CacheEnd** to shut down their connections. Failure to shut down each connection with **CacheEnd** may hang the instance, requiring a restart.

If **CacheSecureStart** is being used, to specify credentials as part of the login, each thread must call **CacheSecureStart** and provide the correct username/password for the connection, since credentials are not shared between the threads. There is a performance penalty within Caché using threads because of the extra code the C compiler has to generate to access thread local storage (which uses direct memory references in non-threaded builds).

A sample program, sampcallint.c, is provided on all platforms where this feature is supported. The vc8 project, and the UNIX® Makefiles, include instructions to build a sample threaded Callin application on the relevant platforms.

2.4.1 Threads and UNIX® Signal Handling

On UNIX®, Caché uses a number of signals. If your application uses the same signals, you should be aware of how Caché deals with them. All signals have a default action specified by the OS. Applications may choose to leave the default action, or can choose to handle or ignore the signal. If the signal is handled, the application may further select which threads will block the signal and which threads will receive the signal. Some signals cannot be blocked, ignored, or handled. Since the default action for many signals is to halt the process, leaving the default action in place is not an option. The following signals cannot be caught or ignored, and terminate the process:

SIGNAL	DISPOSITION
`SIGKILL`	terminate process immediately
`SIGSTOP`	stop process for later resumption

The actions that an application establishes for each signal are process-wide. Whether or not the signal can be delivered to each thread is thread-specific. Each thread may specify how it will deal with signals, independently of other threads. One thread may block all signals, while another thread may allow all signals to be sent to that thread. What happens when a signal is sent to the thread depends on the process-wide handling established for that signal.

2.4.1.1 Caché Signal Processing

Caché integrates with application signal handling by saving application handlers and signal masks, then restoring them at the appropriate time. Caché processes signals in the following ways:

Generated signals

Caché installs its own signal handler for all generated signals. It saves the current (application) signal handler. If the thread catches a generated signal, the Caché signal handler disconnects the thread from Caché, calls the applications signal handling function (if any), then does `pthread_exit`.

Since signal handlers are process-wide, threads not connected to Caché will also go into the Caché handler. If Caché detects that the thread is not connected, it calls the application handler and then does `pthread_exit`.

Synchronous Signals

Caché establishes signal handlers for all synchronous signals, and unblocks these signals for each thread when the thread connects to Caché (see "Synchronous Signals" for details).

Asynchronous Signals

Caché handles all asynchronous signals that would terminate the process (see "Asynchronous Signals" for details).

Save/Restore Handlers

The system saves the signal state when the first thread connects to it. When the last thread disconnects, Caché restores the signal state for every signal that it has handled.

Save/Restore Thread Signal Mask

The thread signal mask is saved on connect, and restored when the thread disconnects.

2.4.1.2 Synchronous Signals

Synchronous signals are generated by the application itself (for example, `SIGSEGV`). Caché establishes signal handlers for all synchronous signals, and unblocks these signals for each thread when it connects to Caché.

Synchronous signals are caught by the thread that generated the signal. If the application has not specified a handler for a signal it has generated (for example, `SIGSEGV`), or if the thread has blocked the signal, then the OS will halt the entire process. If the thread enters the signal handler, that thread may exit cleanly (via `pthread_exit`) with no impact to any other thread. If a thread attempts to return from the handler, the OS will halt the entire process. The following signals cause thread termination:

SIGNAL	DISPOSITION
`SIGABRT`	process abort signal
`SIGBUS`	bus error
`SIGEMT`	EMT instruction
`SIGFPE`	floating point exception
`SIGILL`	illegal instruction
`SIGSEGV`	access violation
`SIGSYS`	bad argument to system call
`SIGTRAP`	trace trap
`SIGXCPU`	CPU time limit exceeded (`setrlimit`)

2.4.1.3 Asynchronous signals

Asynchronous signals are generated outside the application (for example, `SIGALRM`, `SIGINT`, and `SIGTERM`). Caché handles all asynchronous signals that would terminate the process.

Asynchronous signals may be caught by any thread that has not blocked the signal. The system chooses which thread to use. Any signal whose default action is to cause the process to exit must be handled, with at least one thread eligible to receive it, or else it must be specifically ignored.

The application must establish a signal handler for those signals it wants to handle, and must start a thread that does not block those signals. That thread will then be the only one eligible to receive the signal and handle it. Both the handler and the eligible thread must exist before the application makes its first call to **CacheStart**. On the first call to **CacheStart**, the following actions are performed for all asynchronous signals that would terminate the process:

- Caché looks for a handler for these signals. If a handler is found, Caché leaves it in place. Otherwise, Caché sets the signal to `SIG_IGN` (ignore the signal).
- Caché blocks all of these signals for connected threads, whether or not a signal has a handler. Thus, if there is a handler, only a thread that is not connected to Caché can catch the signal.

The following signals are affected by this process:

SIGNAL	DISPOSITION
`SIGALRM`	timer
`SIGCHLD`	blocked by threads
`SIGDANGER`	ignore if unhandled
`SIGHUP`	ignore if unhandled
`SIGINT`	ignore if unhandled
`SIGPIPE`	ignore if unhandled
`SIGQUIT`	ignore if unhandled
`SIGTERM`	If `SIGTERM` is unhandled, Cache will handle it. On receipt of a `SIGTERM` signal, the Cache handler will disconnect all threads and no new connections will be permitted. Handlers for `SIGTERM` are not stacked.
`SIGUSR1`	inter-process communication
`SIGUSR2`	inter-process communication
`SIGVTALRM`	virtual timer
`SIGXFSZ`	Caché asynchronous thread rundown

2.5 Callin Programming Tips

Topics in this section include:

- Tips for All Callin Programs
- Tips for Windows
- Tips for UNIX®, Linux, and Mac OS

2.5.1 Tips for All Callin Programs

Your external program must follow certain rules to avoid corrupting Caché data structures, which can cause a system hang.

- *Limits on the number of open files*

  Your program must ensure that it does not open so many files that it prevents Caché from opening the number of databases or other files it expects to be able to. Normally, Caché looks up the user's open file quota and reserves a

certain number of files for opening databases, allocating the rest for the **Open** command. Depending on the quota, Caché expects to have between 6 and 30 Caché database files open simultaneously, and from 0 to 36 files open with the **Open** command.

- *Maximum Directory Length for Callin Applications*

  The directory containing any Callin application must have a full path that uses fewer than 232 characters. For example, if an application is in the `C:\CacheApps\Accounting\AccountsPayable\` directory, this has 40 characters in it and is therefore valid.

- *Call CacheEnd after CacheStart before halting*

  If your Caché connection was established by a call to **CacheStart**, then you must call **CacheEnd** when you are done with the connection. You can make as many Callin function calls in between as you wish.

  You must call **CacheEnd** even if the connection was broken. The connection can be broken by a call to **CacheAbort** with the **RESJOB** parameter.

  CacheEnd performs cleanup operations which are necessary to prepare for another call to **CacheStart**. Calling **CacheStart** again without calling **CacheEnd** (assuming a broken connection) will return the code CACHE_CONBROKEN.

- *Wait until ObjectScript is done before exiting*

  If you are going to exit your program, you must be certain ObjectScript has completed any outstanding request. Use the Callin function **CacheContext** to determine whether you are within ObjectScript. This call is particularly important in exit handlers and **Ctrl-C** or **Ctrl-Y** handlers. If **CacheContext** returns a non-zero value, you can invoke **CacheAbort**.

- *Maintaining Margins in Callin Sessions*

  While you can set the margin within a Callin session, the margin setting is only maintained for the rest of the current command line. If a program (as with direct mode) includes the line:

  ```
  :Use 0:10 Write x
  ```

  the margin of 10 is established for the duration of the command line.

  Certain calls affect the command line and therefore its margin. These are the calls are annotated as "calls into Caché" in the function descriptions.

- *Avoid signal handling when using CacheStart()*

  CacheStart sets handlers for various signals, which may conflict with signal handlers set by the calling application.

2.5.2 Tips for Windows

These tips apply only to Windows.

- *Limitations on building Callin applications using the cache shared library (cache.dll)*

  If Callin applications are built using the shared library (cache.dll) rather that the static object (cache.obj), users who have large global buffer pools may see the Callin fail to initialize (in CacheStart) with an error:

  ```
  <Cache Startup Error: Mapping shared memory (203)>
  ```

  The explanation for this lies in the behavior of system DLLs loading in Windows. Applications coded in the Win 32 API or with the Microsoft Foundation Classes (the chief libraries that support Microsoft Visual C++ development) need to have the OS load the DLLs for that Windows code as soon as they initialize. These DLLs get loaded from the top of virtual storage (higher addresses), reducing the amount of space left for the heap. On most systems, there are also a number of other DLLs (for example, DLLs supporting the display graphics) that load automatically with each Windows process at locations well above the bottom of the virtual storage. These DLLs have a tendency to request a

specific address space, most commonly 0X10000000 (256MB), chopping off a few hundred megabytes of contiguous memory at the bottom of virtual memory. The result may be that there is insufficient virtual memory space in the Callin executable in which to map the Cache shared memory segment.

2.5.3 Tips for UNIX®, Linux, and Mac OS

These tips apply only to UNIX®, Linux, and Mac OS.

- *Do not disable interrupt delivery on UNIX®*

  UNIX® uses interrupts. Do not prevent delivery of interrupts.

- *Use the correct version of XCode*

  Versions of Caché for Mac OS X (32–bit) previous to 2010.2 were built using the Xcode 2.5 compiler. Callin programs for these versions of Caché must be built using the same compiler. If your development platform is Mac OS X 10.5 (Leopard) or later, you would have to load and use Xcode 2.5 in place of the default Xcode 3.0 compiler.

- *Avoid using reserved signals*

  On UNIX®, Caché uses a number of signals. If possible, application programs linked with Caché should avoid using the following reserved signals:

SIGABRT	SIGDANGER	SIGILL	SIGQUIT	SIGTERM	SIGVTALRM
SIGALRM	SIGEMT	SIGINT	SIGSTOP	SIGTRAP	SIGXCPU
SIGBUS	SIGFPE	SIGKILL	SIGSEGV	SIGUSR1	SIGXFSZ
SIGCHLD	SIGHUP	SIGPIPE	SIGSYS	SIGUSR2	

  If your application uses these signals, you should be aware of how Caché deals with them. See Threads and UNIX® Signal Handling for details.

2.6 Running Sample Programs on Windows

The \dev\cache\callin directory contains source files, header files, and project directories for building Caché Callin applications. These projects provide a simple demonstration of how to use some high level Caché call-in functions.

In order to build these projects, open any of the .vcproj files (for Visual C++ 2005), or .dsp files (for Visual C++ 2003). Double-click on the file, or run your Visual C++ application and select `File>Open>Project/Solution` to open the project file.

Note: You can run call-in programs on Windows 2000, but you have to compile them on Windows XP or newer, since Visual Studio 2008 and the Windows 2008 SDK only go back to Windows XP. The Visual Studio 2008 redistributables are supported on Windows 2000, but there does not appear to be a compatible compiler that is supported on Windows 2000.

The shdir.c file has been already initialized with the path to your Caché mgr directory. For a default installation, the shdir.c file will look like this:

```
char shdir[256] = "c:\\cachesys\\mgr";
```

The Callin interface provides the CACHESETDIR entry point to dynamically set the name of the manager directory at runtime. The shared library version of cache requires the use of this interface to find the installation's manager's directory.

Two sample C programs are provided. The sampcallin.c program is the standard Callin application example, and sampcallint.c is the thread-safe Callin application example.

There are two projects for sampcallin.c and a project for sampcallint.c. These projects are:

- callin — builds a statically linked Callin application using cache.obj.
- callinsh — builds a dynamically linked Callin application using cache.dll.
- callint — builds a dynamically linked thread-safe Callin application, using cachet.dll.

After each of the projects is built, it may be run in the Visual C++ environment.

When a project is built from the cache shared library, using cache.dll, the location of cache.dll must be defined in the user's PATH environment variable, except when the file is located in the current directory.

2.7 Running Sample Programs on UNIX® and Linux

The directory dev/cache/callin/samples contains a complete Makefile to build Callin samples. This replaces the clink file found in previous releases.

A shared library version of cache is now provided in addition to the cache object file. The UNIX® Makefiles build two Callin sample applications: one using the cache object, and one using the libcache shared library.

Run make in the dev/cache/callin/samples directory. The supplied Makefile will build a cache using the czf interface, a standard Callin application, and a shared library Callin application.

The file shdir.c is set to the appropriate value during installation, so no editing is required.

The Callin interface provides the CACHESETDIR entry point to dynamically set the name of the manager directory at runtime.

Using Makefiles on UNIX®

The UNIX® Makefiles for building Callin samples and customer Callin programs are run by the **make** command. **make** automatically finds the file called Makefile in the current directory. Thus, running **make** in the samples directory produces a sample Callin executable.

When invoking make, use the *SRC* variable to specify the name of the source program. The default is *sampcallin*. To change the name of the source file being built, override the *SRC* variable on the command line. For example, with a Callin program called mycallin.c, the command is:

```
make SRC=mycallin
```

Setting Permissions for Callin Executables on UNIX®

Caché executables, files, and resources such as shared memory and operating system messages, are owned by a user selected at installation time (the installation owner) and a group with a default name of cacheusr (you can choose a different name at installation time). These files and resources are only accessible to processes that either have this user ID or belong to this group. Otherwise, attempting to connect to Caché results in protection errors from the operating system (usually specifying that access is denied); this occurs prior to establishing any connection with Caché.

A Callin program can only run if its effective group ID is cacheusr. To meet this condition, one of the following must be true:

- The program is run by a user in the cacheusr group (or an alternate run-as group if it was changed from cacheusr to something else).

- The program sets its effective user or group by manipulating its uid or gid file permissions (using the UNIX® **chgrp** and **chmod** commands).

3

Using the Callin Functions

This section provides a quick summary of the Callin functions, with links to the full description of each function. The following categories are discussed:

- Process Control

  These functions start and stop a Callin session, and control various settings associated with the session.

- Functions and Routines

  These functions execute function or routine calls. Stack functions are provided for pushing function or routine references.

- Transactions and Locking

  These functions execute the standard Caché transaction commands (TSTART, TCOMMIT, and TROLLBACK) and the LOCK command.

- Managing Objects

  These functions manipulate the Oref counter, perform method calls, and get or set property values. Stack functions are also included for Orefs, method references, and property names.

- Managing Globals

  These functions call into Caché to manipulate globals. Functions are provided to push globals onto the argument stack.

- Managing Strings

  These functions translate strings from one form to another, and push or pop string arguments.

- Managing Simple Datatypes

  These stack functions are used to push and pop arguments that have int, double, $list, or pointer values.

The following sections discuss the individual functions in more detail.

3.1 Process Control

These functions start and stop a Callin session, control various settings associated with the session, and provide a high-level interface for executing ObjectScript commands and expressions.

3.1.1 Session Control

These functions start and stop a Callin session, and control various settings associated with the session.

Table 3–1: Session control functions

CacheAbort	Tells Caché to terminate the current request.
CacheChangePasswordA[W][H]	Changes the user's password if Caché authentication is used. Must be called before a Callin session is initialized.
CacheContext	Returns an integer indicating whether you are in a **$ZF** callback session, in the Caché side of a Callin call, or in the user program side.
CacheCtrl	Determines whether or not Caché ignores **CTRL-C**.
CacheEnd	Terminates a Caché session and, if necessary, cleans up a broken connection. (Calls into Caché).
CacheEndAll	Disconnects all Callin threads and waits until they terminate.
CacheOflush	Flushes any pending output.
CachePromptA[W][H]	Returns a string that would be the programmer prompt.
CacheSetDir	Dynamically sets the name of the manager's directory (CacheSys\Mgr) at runtime. On Windows, the shared library version of Caché requires this function.
CacheSignal	Reports a signal detected by the user program to Caché for handling.
CacheSecureStartA[W][H]	Initiates a Caché process.
CacheStartA[W][H]	(Deprecated. Use **CacheSecureStart** instead) Initiates a Caché process.

3.1.2 Running ObjectScript

These functions provide a high-level interface for executing ObjectScript commands and expressions.

Table 3–2: ObjectScript command functions

CacheExecuteA[W][H]	Executes an ObjectScript command. (Calls into Caché).
CacheEvalA[W][H]	Evaluates an ObjectScript expression. (Calls into Caché).
CacheConvert	Returns the value of the Caché expression returned by **CacheEval**.
CacheType	Returns the datatype of an item returned by **CacheEval**.
CacheErrorA[W][H]	Returns the most recent error message, its associated source string, and the offset to where in the source string the error occurred.
CacheErrxlateA[W][H]	Returns the Caché error string associated with error number returned from a Callin function.

3.2 Functions and Routines

These functions call into Caché to perform function or routine calls. Functions are provided to push function or routine references onto the argument stack.

Table 3–3: Functions for performing function and routine calls

CacheDoFun	Perform a routine call (special case). (Calls into Caché).
CacheDoRtn	Perform a routine call. (Calls into Caché).
CacheExtFun	Perform an extrinsic function call. (Calls into Caché).
CachePop	Pops a value off argument stack.
CacheUnPop	Restores the stack entry from **CachePop**
CachePushFunc[**W**][**H**]	Pushes an extrinsic function reference onto the argument stack.
CachePushFuncX[**W**][**H**]	Push an extended function reference onto argument stack
CachePushRtn[**W**][**H**]	Push a routine reference onto argument stack
CachePushRtnX[**W**][**H**]	Push an extended routine reference onto argument stack

3.3 Transactions and Locking

These functions execute the standard Caché transaction commands (TSTART, TCOMMIT, and TROLLBACK) and the LOCK command.

3.3.1 Transactions

The following functions execute the standard Caché transaction commands.

Table 3–4: Transaction functions

CacheTCommit	Executes a Caché TCommit command.
CacheTLevel	Returns the current nesting level ($TLEVEL) for transaction processing.
CacheTRollback	Executes a Caché TRollback command.
CacheTStart	Executes a Caché TStart command.

3.3.2 Locking

These functions execute various forms of the Cache LOCK command. Functions are provided to push lock names onto the argument stack for use by the CacheAcquireLock function.

Table 3–5: Locking functions

CacheAcquireLock	Executes a Caché LOCK command.
CacheReleaseAllLocks	Performs an argumentless Cache LOCK command to remove all locks currently held by the process.
CacheReleaseLock	Executes a Cache LOCK — command to decrement the lock count for the specified lock name.
CachePushLock[**W**][**H**]	Initializes a CacheAcquireLock command by pushing the lock name on the argument stack.
CachePushLockX[**W**][**H**]	Initializes a CacheAcquireLock command by pushing the lock name and an environment string on the argument stack.

3.4 Managing Objects

These functions call into Caché to manipulate the Oref counter, perform method calls, and get or set property values. Stack functions are also included for Orefs, method references, and property names.

3.4.1 Orefs

Table 3–6: Oref functions

CacheCloseOref	Decrement the reference counter for an OREF. (Calls into Caché).
CacheIncrementCountOref	Increment the reference counter for an OREF
CachePopOref	Pop an OREF off argument stack
CachePushOref	Push an OREF onto argument stack

3.4.2 Methods

Table 3–7: Method functions

CacheInvokeMethod	Perform an instance method call. (Calls into Caché).
CachePushMethod[**W**][**H**]	Push an instance method reference onto argument stack
CacheInvokeClassMethod	Perform a class method call. (Calls into Caché).
CachePushClassMethod[**W**][**H**]	Push a class method reference onto argument stack

3.4.3 Properties

Table 3–8: Property functions

CacheGetProperty	Obtain the value for a property. (Calls into Caché).
CacheSetProperty	Store the value for a property. (Calls into Caché).
CachePushProperty[**W**][**H**]	Push a property name onto argument stack

3.5 Managing Globals

These functions call into Caché to manipulate globals. Functions are provided to push globals onto the argument stack.

Table 3–9: Functions for managing globals

CacheGlobalGet	Obtains the value of the global reference defined by **CachePushGlobal**[**W**][**H**] and any subscripts. The node value is pushed onto the argument stack.
CacheGlobalGetBinary	Obtains the value of the global reference like **CacheGlobalGet**, and also tests to make sure that the result is a binary string that will fit in the provided buffer.
CacheGlobalSet	Stores the value of the global reference. The node value must be pushed onto the argument stack before this call.
CacheGlobalData	Performs a $Data on the specified global.
CacheGlobalIncrement	Performs a $Increment and returns the result on top of the stack.
CacheGlobalKill	Performs a ZKILL on a global node or tree.
CacheGlobalOrder	Performs a $Order on the specified global.
CacheGlobalQuery	Performs a $Query on the specified global.
CacheGlobalRelease	Releases ownership of a retained global buffer, if one exists.
CachePushGlobal[**W**][**H**]	Pushes a global name onto argument stack
CachePushGlobalX[**W**][**H**]	Pushes an extended global name onto argument stack

3.6 Managing Strings

These functions translate strings from one form to another, and push or pop string arguments.

3.6.1 Long String Functions

Caché long string functions may be used for both long strings and standard strings. Functions are provided for local 8-bit encoding, 2–byte Unicode, and 4–byte Unicode.

Table 3–10: Long string functions

CacheCvtExStrInA[**W**][**H**]	Translates a string with specified external character set encoding to the character string encoding used internally by Caché.
CacheCvtExStrOutA[**W**][**H**]	Translates a string from the character string encoding used internally in Caché to a string with the specified external character set encoding.
CacheExStrKill	Releases the storage associated with a long string.
CacheExStrNew[**W**][**H**]	Allocates the requested amount of storage for a long string, and fills in the EXSTR structure with the length and a pointer to the value field of the structure.
CachePopExStrCvtW[**H**]	Pops a string off the argument stack and translates it to a Unicode string.
CachePushExStrCvtW[**H**]	Converts a Unicode string to local 8–bit encoding and pushes it onto the argument stack.
CachePopExStr[**W**][**H**]	Pops a value off argument stack and converts it to a string of the desired type.
CachePushExStr[**W**][**H**]	Pushes a string onto the argument stack

3.6.2 Standard String Functions

The following functions deal with standard Caché strings (limited to 32K). Functions are provided for local 8-bit encoding, 2–byte Unicode, and 4–byte Unicode.

Table 3–11: Standard string functions

CacheCvtInA[**W**][**H**]	Translates a string with the specified external character set encoding to the character string encoding used internally in Caché.
CacheCvtOutA[**W**][**H**]	Translates a string from the character string encoding used internally in Caché to a string with the specified external character set encoding.
CachePopStr[**W**][**H**]	Pops a value off argument stack and converts it to a string of the desired type.
CachePushStr[**W**][**H**]	Pushes a string onto argument stack
CachePushCvtW[**H**]	Translates a Unicode string to local and pushes it onto argument stack
CachePopCvtW[**H**]	Pops a value off argument stack and translates it into the desired string type.

3.7 Managing Other Datatypes

These functions are used to push and pop argument values with datatypes such as int, double, $list, or pointer, and to return the position of specified bit values within a bitstring.

Table 3–12: Other datatype functions

CachePushInt	Push an integer onto argument stack

CachePopInt	Pop a value off argument stack and convert it to an integer
CachePushInt64	Push a 64–bit (long long) value onto argument stack
CachePopInt64	Pop a value off argument stack and convert it to a 64–bit (long long) value
CachePushDbl	Push a Caché double onto argument stack
CachePushIEEEDbl	Push an IEEE double onto argument stack.
CachePopDbl	Pops value off argument stack and converts it to a double
CachePushList	Translates and pushes a **$LIST** object onto argument stack
CachePopList	Pops a **$LIST** object off argument stack and translates it
CachePushPtr	Pushes a pointer value onto argument stack
CachePopPtr	Pops a pointer value off argument stack
CachePushUndef	Pushes an Undefined value that is interpreted as an omitted function argument.
CacheBitFind[**B**]	Returns the position of specified bit values within a bitstring. Similar to Caché $BITFIND.

4

Callin Function Reference

This reference chapter contains detailed descriptions of all Caché Callin functions, arranged in alphabetical order. For an introduction to the Callin functions organized by function, see Using the Callin Functions.

Note: Caché Callin functions that operate on strings have both 8-bit and Unicode versions. These functions use a suffix character to indicate the type of string that they handle:

- Names with an "A" suffix or no suffix at all (for example,**CacheEvalA** or **CachePopStr**) are versions for 8-bit character strings.
- Names with a "W" suffix (for example,**CacheEvalW** or **CachePopStrW**) are versions for Unicode character strings on platforms that use 2–byte Unicode characters.
- Names with an "H" suffix (for example,**CacheEvalH** or **CachePopStrH**) are versions for Unicode character strings on platforms that use 4–byte Unicode characters.

For convenience, the different versions of each function are listed together here. For example, **CacheEvalA**[**W**][**H**] or **CachePopStr**[**W**][**H**].

4.1 Alphabetical Function List

This section contains an alphabetical list of all Callin functions with a brief description of each function and links to detailed descriptions.

- **CacheAbort** — Tells Caché to cancel the current request being processed on the Caché side, when it is convenient to do so.
- **CacheAcquireLock** — Executes a Cache LOCK command. The lock reference should already be set up with **CachePushLockX**[**W**][**H**].
- **CacheChangePasswordA**[**W**][**H**] — Changes the user's password if Caché authentication is used (not valid for other forms of authentication).
- **CacheBitFind**[**B**] — Returns the position of specified bit values within a bitstring (similar to Caché $BITFIND).
- **CacheCloseOref** — Decrements the system reference counter for an OREF.
- **CacheContext** — Returns `true` if there is a request currently being processed on the Caché side of the connection when using an external Callin program.
- **CacheConvert** — Converts the value returned by **CacheEvalA[W][H]** into proper format and places in address specified in its return value.

- **CacheCtrl** — Determines whether or not Caché ignores **CTRL-C**.
- **CacheCvtExStrInA**[**W**][**H**] — Translates a string with specified external character set encoding to the local 8-bit character string encoding used internally only in 8-bit versions of Caché.
- **CacheCvtExStrOutA**[**W**][**H**] — Translates a string from the local 8-bit character string encoding used internally in the Caché 8-bit product to a string with the specified external character set encoding. (This is only available with 8-bit versions of Caché.)
- **CacheCvtInA**[**W**][**H**] — Translates string with specified external character set encoding to the local 8-bit character string encoding (used internally only in 8-bit versions of Caché) or the Unicode character string encoding (used internally in Unicode versions of Caché).
- **CacheCvtOutA**[**W**][**H**] — Translates a string from the local 8-bit character string encoding used internally in the Caché 8-bit product to a string with the specified external character set encoding. (This is only available with 8-bit versions of Caché.)
- **CacheDoFun** — Performs a routine call (special case).
- **CacheDoRtn** — Performs a routine call.
- **CacheEnd** — Terminates a Caché process. If there is a broken connection, it also performs clean-up operations.
- **CacheEndAll** — Disconnects all Callin threads and waits until they terminate.
- **CacheErrorA**[**W**][**H**] — Returns the most recent error message, its associated source string, and the offset to where in the source string the error occurred.
- **CacheErrxlateA**[**W**][**H**] — Translates an integer error code into a Cache error string.
- **CacheEvalA**[**W**][**H**] — Evaluates a string as if it were a Caché expression and places the return value in memory for further processing by **CacheType** and **CacheConvert**.
- **CacheExecuteA**[**W**][**H**] — Executes a command string as if it were typed at the Caché programmer prompt.
- **CacheExStrKill** — Releases the storage associated with an EXSTR string.
- **CacheExStrNew**[**W**][**H**] — Allocates the requested amount of storage for a string, and fills in the EXSTR structure with the length and a pointer to the value field of the structure.
- **CacheExtFun** — Performs an extrinsic function call where the return value is pushed onto the argument stack.
- **CacheGetProperty** — Obtains the value of the property defined by **CachePushProperty**[**W**][**H**]. The value is pushed onto the argument stack.
- **CacheGlobalData** — Performs a $Data on the specified global.
- **CacheGlobalGet** — Obtains the value of the global reference defined by **CachePushGlobal**[**W**][**H**] and any subscripts. The node value is pushed onto the argument stack.
- **CacheGlobalIncrement** — Performs a $INCREMENT and returns the result on top of the stack.
- **CacheGlobalKill** — Performs a ZKILL on a global node or tree.
- **CacheGlobalOrder** — Performs a $Order on the specified global.
- **CacheGlobalQuery** — Performs a $Query on the specified global.
- **CacheGlobalRelease** — Release ownership of a retained global buffer, if one exists.
- **CacheGlobalSet** — Stores the value of the global reference defined by **CachePushGlobal**[**W**][**H**] and any subscripts. The node value must be pushed onto the argument stack before this call.
- **CacheIncrementCountOref** — Increments the system reference counter for an OREF.

- **CacheInvokeClassMethod** — Executes the class method call defined by **CachePushClassMethod**[**W**][**H**] and any arguments. The return value is pushed onto the argument stack.
- **CacheInvokeMethod** — Executes the instance method call defined by **CachePushMethod**[**W**][**H**] and any arguments pushed onto the argument stack.
- **CacheOflush** — Flushes any pending output.
- **CachePop** — Pops a value off argument stack.
- **CachePopCvtW**[**H**] — Pops a local 8-bit string off argument stack and translates it to Unicode. Identical to **CachePopStr**[**W**][**H**] for Unicode versions.
- **CachePopDbl** — Pops a value off argument stack and converts it to a double.
- **CachePopExStr**[**W**][**H**] — Pops a value off argument stack and converts it to a long string.
- **CachePopExStrCvtW**[**H**] — Pops a value off argument stack and converts it to a long Unicode string.
- **CachePopInt** — Pops a value off argument stack and converts it to an integer.
- **CachePopInt64** — Pops a value off argument stack and converts it to a 64-bit (long long) number.
- **CachePopList** — Pops a $LIST object off argument stack and converts it. String elements are copied or translated as appropriate depending on whether this is a Unicode or 8-bit version.
- **CachePopOref** — Pops an OREF off argument stack.
- **CachePopPtr** — Pops a pointer off argument stack in internal format.
- **CachePopStr**[**W**][**H**] — Pops a value off argument stack and converts it to a string.
- **CachePromptA**[**W**][**H**] — Returns a string that would be the programmer prompt.
- **CachePushClassMethod**[**W**][**H**] — Pushes a class method reference onto the argument stack.
- **CachePushCvtW**[**H**] — Translates a Unicode string to local 8-bit and pushes it onto the argument stack. Identical to **CachePushStr**[**W**][**H**] for Unicode versions.
- **CachePushDbl** — Pushes a Caché double onto the argument stack.
- **CachePushExStr**[**W**][**H**] — Pushes a long string onto the argument stack.
- **CachePushExStrCvtW**[**H**] — Translates a Unicode string to local 8-bit and pushes it onto the argument stack.
- **CachePushFunc**[**W**][**H**] — Pushes an extrinsic function reference onto the argument stack.
- **CachePushFuncX**[**W**][**H**] — Pushes an extended extrinsic function reference onto the argument stack.
- **CachePushGlobal**[**W**][**H**] — Pushes a global reference onto the argument stack.
- **CachePushGlobalX**[**W**][**H**] — Pushes an extended global reference onto the argument stack.
- **CachePushIEEEDbl** — Pushes an IEEE double onto the argument stack.
- **CachePushInt** — Pushes an integer onto the argument stack.
- **CachePushInt64** — Pushes a 64-bit (long long) number onto the argument stack.
- **CachePushList** — Converts a $LIST object and pushes it onto the argument stack.
- **CachePushLock**[**W**][**H**] — Initializes a **CacheAcquireLock** command by pushing the lock name on the argument stack.
- **CachePushLockX**[**W**][**H**] — Initializes a **CacheAcquireLock** command by pushing the lock name and an environment string on the argument stack.
- **CachePushMethod**[**W**][**H**] — Pushes an instance method reference onto the argument stack.

- **CachePushOref** — Pushes an OREF onto the argument stack.
- **CachePushProperty**[**W**][**H**] — Pushes a property reference onto the argument stack.
- **CachePushPtr** — Pushes a pointer onto the argument stack in internal format.
- **CachePushRtn**[**W**][**H**] — Pushes a routine reference onto the argument stack.
- **CachePushRtnX**[**W**][**H**] — Pushes an extended routine reference onto the argument stack.
- **CachePushStr**[**W**][**H**] — Pushes a byte string onto the argument stack.
- **CachePushExStrCvtW**[**H**] — Converts a Unicode string to local 8–bit encoding and pushes it onto the argument stack.
- **CachePushUndef** — pushes an Undefined value that is interpreted as an omitted function argument.
- **CacheReleaseAllLocks** — Performs an argumentless Cache LOCK command to remove all locks currently held by the process.
- **CacheReleaseLock** — Executes a Cache LOCK command to decrement the lock count for the specified lock name. This command will only release one incremental lock at a time.
- **CacheSecureStartA**[**W**][**H**] — Calls into Cache to set up a Cache process.
- **CacheSetDir** — Dynamically sets the name of the manager's directory at runtime.
- **CacheSetProperty** — Stores the value of the property defined by **CachePushProperty**[**W**][**H**].
- **CacheSignal** — Passes on signals caught by user's program to Caché.
- **CacheSPCReceive** — Receive single-process-communication message.
- **CacheSPCSend** — Send a single-process-communication message.
- **CacheStartA**[**W**][**H**] — Calls into Caché to set up a Caché process.
- **CacheTCommit** — Executes a Cache TCommit command.
- **CacheTLevel** — Returns the current nesting level ($TLEVEL) for transaction processing.
- **CacheTRollback** — Executes a Cache TRollback command.
- **CacheTStart** — Executes a Cache TStart command.
- **CacheType** — Returns the native type of the item returned by **CacheEvalA**[**W**][**H**], as the function value.
- **CacheUnPop** — Restores the stack entry from **CachePop**.

4.2 CacheAbort

```
int CacheAbort(unsigned long type)
```

Arguments

type	Either of the following predefined values that specify how the termination occurs: • `CACHE_CTRLC` — Interrupts the Caché processing as if a **CTRL-C** had been processed (regardless of whether **CTRL-C** has been enabled with **CacheCtrl**). A connection to Caché remains. • `CACHE_RESJOB` — Terminates the Callin connection. You must then call **CacheEnd** and then **CacheStart** to reconnect to Caché.

Description

Tells Caché to cancel the current request being processed on the Caché side, when it is convenient to do so. This function is for use if you detect some critical event in an AST (asynchronous trap) or thread running on the Callin side. (You can use **CacheContext** to determine if there is a Caché request currently being processed.) Note that this only applies to Callin programs that use an AST or separate thread.

Return Values for CacheAbort

CACHE_BADARG	The termination type is invalid.
CACHE_CONBROKEN	Connection has been broken.
CACHE_NOCON	No connection has been established.
CACHE_NOTINCACHE	The Callin partner is not in Caché at this time.
CACHE_SUCCESS	Connection formed.

Example

```
rc = CacheAbort(CACHE_CTRLC);
```

4.3 CacheAcquireLock

```
int CacheAcquireLock(int nsub, int flg, int tout, int * rval)
```

Arguments

nsub	Number of subscripts in the lock reference.
flg	Modifiers to the lock command. Valid values are one or both of CACHE_INCREMENTAL_LOCK and CACHE_SHARED_LOCK.
tout	Number of seconds to wait for the lock command to complete. Negative for no timeout. 0 means return immediately if the lock is not available, although a minimum timeout may be applied if the lock is mapped to a remote system.
rval	Optional pointer to an int return value: success = 1, failure = 0.

Description

Executes a Cache LOCK command. The lock reference should already be set up with **CachePushLock**.

Return Values for CacheAcquireLock

CACHE_FAILURE	An unexpected error has occurred.
CACHE_SUCCESS	Successfully called the LOCK command (but the *rval* parameter must be examined to determine if the lock succeeded).
CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_ERARGSTACK	Argument stack overflow.

4.4 CacheBitFind

```
int CacheBitFind(int strlen, unsigned short *bitstr, int newlen, int srch, int revflg)
```

Arguments

strlen	Data length of the bitstring.
bitstr	Pointer to a Unicode bitstring.
newlen	`0` to start at the beginning, otherwise 1–based starting position
srch	The bit value (`0` or `1`) to search for within the bitstring.
revflg	Specifies the search direction: `1` — Search forward (left to right) from the position indicated by *newlen*. `0` — Search backward from the position indicated by *newlen*.

Description

Returns the bit position (1-based) of the next bit within bitstring *bitstr* that has the value specified by *srch*. The direction of the search is indicated by *revflg*. Returns `0` if there are no more bits of the specified value in the specified direction.

This function is similar to Caché $BITFIND (also see "General Information on Bitstring Functions").

Return Values for CacheBitFind

CACHE_SUCCESS	The operation was successful.

4.5 CacheBitFindB

```
int CacheBitFindB(int strlen, unsigned char *bitstr, int newlen, int srch, int revflg)
```

Arguments

strlen	Data length of the bitstring.
bitstr	Pointer to a bitstring.
newlen	`0` to start at the beginning, otherwise 1–based starting position.
srch	The bit value (`0` or `1`) to search for within the bitstring.
revflg	Specifies the search direction: `1` — Search forward (left to right) from the position indicated by *newlen*. `0` — Search backward from the position indicated by *newlen*.

Description

Returns the bit position (1-based) of the next bit within bitstring *bitstr* that has the value specified by *srch*. The direction of the search is indicated by *revflg*. Returns `0` if there are no more bits of the specified value in the specified direction.

This function is similar to Caché $BITFIND (also see "General Information on Bitstring Functions").

Return Values for CacheBitFindB

CACHE_SUCCESS	The operation was successful.

4.6 CacheChangePasswordA

Variants: **CacheChangePasswordW**, **CacheChangePasswordH**

```
int CacheChangePasswordA(CACHE_ASTRP username, CACHE_ASTRP oldpassword, CACHE_ASTRP newpassword)
```

Arguments

username	Username of the user whose password must be changed.
oldpassword	User's old password.
newpassword	New password.

Description

This function can change the user's password if Caché authentication or delegated authentication is used. It is not valid for LDAP, Kerberos, or other forms of authentication. It must be called before a Callin session is initialized. A typical use would be to handle a `CACHE_CHANGEPASSWORD` error from **CacheSecureStart**. In such a case **CacheChangePassword** would be called to change the password, then **CacheSecureStart** would be called again.

Return Values for CacheChangePasswordA

CACHE_FAILURE	An unexpected error has occurred.
CACHE_SUCCESS	Password changed.

4.7 CacheChangePasswordH

Variants: **CacheChangePasswordA**, **CacheChangePasswordW**

```
int CacheChangePasswordH(CACHEHSTRP username, CACHEHSTRP oldpassword, CACHEHSTRP newpassword)
```

Arguments

username	Username of the user whose password must be changed.
oldpassword	User's old password.
newpassword	New password.

Description

This function can change the user's password if Caché authentication or delegated authentication is used. It is not valid for LDAP, Kerberos, or other forms of authentication. It must be called before a Callin session is initialized. A typical use would be to handle a `CACHE_CHANGEPASSWORD` error from **CacheSecureStart**. In such a case **CacheChangePassword** would be called to change the password, then **CacheSecureStart** would be called again.

Return Values for CacheChangePasswordH

CACHE_FAILURE	An unexpected error has occurred.
CACHE_SUCCESS	Password changed.

4.8 CacheChangePasswordW

Variants: **CacheChangePasswordA**, **CacheChangePasswordH**

```
int CacheChangePasswordW(CACHEWSTRP username, CACHEWSTRP oldpassword, CACHEWSTRP newpassword)
```

Arguments

username	Username of the user whose password must be changed.
oldpassword	User's old password.
newpassword	New password.

Description

This function can change the user's password if Caché authentication or delegated authentication is used. It is not valid for LDAP, Kerberos, or other forms of authentication. It must be called before a Callin session is initialized. A typical use would be to handle a `CACHE_CHANGEPASSWORD` error from **CacheSecureStart**. In such a case **CacheChangePassword** would be called to change the password, then **CacheSecureStart** would be called again.

Return Values for CacheChangePasswordW

CACHE_FAILURE	An unexpected error has occurred.
CACHE_SUCCESS	Password changed.

4.9 CacheCloseOref

```
int CacheCloseOref(unsigned int oref)
```

Arguments

oref	Object reference.

Description

Decrements the system reference counter for an OREF.

Return Values for CacheCloseOref

CACHE_ERBADOREF	Invalid OREF.
CACHE_SUCCESS	The operation was successful.

4.10 CacheContext

```
int CacheContext()
```

Description

Returns an integer as the function value.

If you are using an external Callin program (as opposed to a module that was called from a **$ZF** function) and your program employs an AST or separate thread, then **CacheContext** tells you if there is a request currently being processed on the Caché side of the connection. This information is needed to decide if you must return to Caché to allow processing to complete.

Return Values for CacheContext

`-1`	Created in Caché via a **$ZF** callback.
0	No connection or not in Caché at the moment.
1	In Caché via an external (i.e., not **$ZF**) connection. An asynchronous trap (AST), such as an exit-handler, would need to return to Caché to allow Caché to complete processing.

Note: The information about whether you are in a **$ZF** function from a program or an AST is needed because, if you are in an AST, then you need to return to Caché to allow processing to complete.

Example

```
rc = CacheContext();
```

4.11 CacheConvert

```
int CacheConvert(unsigned long type, void * rbuf)
```

Arguments

type	The #define'd type, with valid values listed below.
rbuf	Address of a data area of the proper size for the data type. If the type is CACHE_ASTRING, *rbuf* should be the address of a CACHE_ASTR structure that will contain the result, and the *len* element in the structure should be filled in to represent the maximum size of the string to be returned (in characters). Similarly, if the type is CACHE_WSTRING, *rbuf* should be the address of a CACHEWSTR structure whose *len* element has been filled in to represent the maximum size (in characters).

Description

Converts the value returned by **CacheEval** into proper format and places in address specified in its return value (listed below as *rbuf*).

Valid values of *type* are:

- CACHE_ASTRING — 8-bit character string.
- CACHE_CHAR — 8-bit signed integer.

- CACHE_DOUBLE — 64-bit floating point.
- CACHE_FLOAT — 32-bit floating point.
- CACHE_INT — 32-bit signed integer.
- CACHE_INT2 — 16-bit signed integer.
- CACHE_INT4 — 32-bit signed integer.
- CACHE_INT8 — 64-bit signed integer.
- CACHE_UCHAR — 8-bit unsigned integer.
- CACHE_UINT — 32-bit unsigned integer.
- CACHE_UINT2 — 16-bit unsigned integer.
- CACHE_UINT4 — 32-bit unsigned integer.
- CACHE_UINT8 — 64-bit unsigned integer.
- CACHE_WSTRING — Unicode character string.

Return Values for CacheConvert

CACHE_BADARG	Type is invalid.
CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_ERSYSTEM	Either ObjectScript generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
CACHE_FAILURE	An unexpected error has occurred.
CACHE_NOCON	No connection has been established.
CACHE_NORES	No result whose type can be returned (no call to **CacheEvalA** preceded this call).
CACHE_RETTRUNC	Success, but the type CACHE_ASTRING, CACHE_INT8, CACHE_UINT8 and CACHE_WSTRING resulted in a value that would not fit in the space allocated in *retval*. For CACHE_INT8 and CACHE_UINT8, this means that the expression resulted in a floating point number that could not be normalized to fit within 64 bits.
CACHE_STRTOOLONG	String is too long.
CACHE_SUCCESS	Value returned by last **CacheEval** converted successfully.

Note: Caché may perform division when calculating the return value for floating point types, CACHE_FLOAT and CACHE_DOUBLE, which have decimal parts (including negative exponents), as well as the 64-bit integer types (CACHE_INT8 and CACHE_UINT8). Therefore, the returned result may not be identical in value to the original. CACHE_ASTRING, CACHE_INT8, CACHE_UINT8 and CACHE_WSTRING can return the status CACHE_RETTRUNC.

Example

```
CACHE_ASTR retval;
/* define variable retval */

retval.len = 20;
/* maximum return length of string */

rc = CacheConvert(CACHE_ASTRING,&retval);
```

4.12 CacheCtrl

```
int CacheCtrl(unsigned long flags)
```

Arguments

flags	Either of two #define'd values specifying how Caché handles certain keystrokes.

Description

Determines whether or not Caché ignores `CTRL-C`. *flags* can have bit state values of

- `CACHE_DISACTRLC` — Caché ignores `CTRL-C`.
- `CACHE_ENABCTRLC` — Default if function is not called, unless overridden by a **BREAK** or an **OPEN** command. In Caché, `CTRL-C` generates an <INTERRUPT>.

Return Values for CacheCtrl

CACHE_FAILURE	Returns if called from a **$ZF** function (rather than from within a Callin executable).
CACHE_SUCCESS	Control function performed.

Example

```
rc = CacheCtrl(CACHE_ENABCTRLC);
```

4.13 CacheCvtExStrInA

Variants: **CacheCvtExStrInW**, **CacheCvtExStrInH**

```
int CacheCvtExStrInA(CACHE_EXSTRP src, CACHE_ASTRP tbl, CACHE_EXSTRP res)
```

Arguments

src	Address of a CACHE_EXSTRP variable that contains the string to be converted.
tbl	The name of the I/O translation table to use to perform the translation (a null string indicates that the default process I/O translation table name should be used).
res	Address of a CACHE_EXSTRP variable that will contain the result.

Description

Translates a string with specified external character set encoding to the local 8-bit character string encoding used internally only in 8-bit versions of Caché.

Return Values for CacheCvtExStrInA

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_ERRUNIMPLEMENTED	Not available for Unicode.
CACHE_ERVALUE	The specified I/O translation table name was undefined or did not have an input component.
CACHE_ERXLATE	Input string could not be translated using the specified I/O translation table.
CACHE_NOCON	No connection has been established.
CACHE_RETTRUNC	Result was truncated because result buffer was too small.
CACHE_FAILURE	Error encountered while trying to build translation data structures (probably not enough partition memory).
CACHE_SUCCESS	Translation completed successfully.

4.14 CacheCvtExStrInW

Variants: **CacheCvtExStrInA**, **CacheCvtExStrInH**

```
int CacheCvtExStrInW(CACHE_EXSTRP src, CACHEWSTRP tbl, CACHE_EXSTRP res)
```

Arguments

src	Address of a CACHE_EXSTRP variable that contains the string to be converted.
tbl	The name of the I/O translation table to use to perform the translation (a null string indicates that the default process I/O translation table name should be used).
res	Address of a CACHE_EXSTRP variable that will contain the result.

Description

Translates a string with specified external character set encoding to the 2–byte Unicode character string encoding used internally in Unicode versions of Caché.

Return Values for CacheCvtExStrInW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_ERRUNIMPLEMENTED	Not available for 8–bit systems.
CACHE_ERVALUE	The specified I/O translation table name was undefined or did not have an input component.
CACHE_ERXLATE	Input string could not be translated using the specified I/O translation table.
CACHE_NOCON	No connection has been established.
CACHE_RETTRUNC	Result was truncated because result buffer was too small.
CACHE_FAILURE	Error encountered while trying to build translation data structures (probably not enough partition memory).
CACHE_SUCCESS	Translation completed successfully.

4.15 CacheCvtExStrInH

Variants: **CacheCvtExStrInA**, **CacheCvtExStrInW**

```
int CacheCvtExStrInH(CACHE_EXSTRP src, CACHEWSTRP tbl, CACHE_EXSTRP res)
```

Arguments

src	Address of a CACHE_EXSTRP variable that contains the string to be converted.
tbl	The name of the I/O translation table to use to perform the translation (a null string indicates that the default process I/O translation table name should be used).
res	Address of a CACHE_EXSTRP variable that will contain the result.

Description

Translates a string with specified external character set encoding to the 4–byte Unicode character string encoding used internally in Unicode versions of Caché.

Return Values for CacheCvtExStrInH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_ERRUNIMPLEMENTED	Not available for 8–bit systems.
CACHE_ERVALUE	The specified I/O translation table name was undefined or did not have an input component.
CACHE_ERXLATE	Input string could not be translated using the specified I/O translation table.
CACHE_NOCON	No connection has been established.
CACHE_RETTRUNC	Result was truncated because result buffer was too small.
CACHE_FAILURE	Error encountered while trying to build translation data structures (probably not enough partition memory).
CACHE_SUCCESS	Translation completed successfully.

4.16 CacheCvtExStrOutA

Variants: **CacheCvtExStrOutW**, **CacheCvtExStrOutH**

```
int CacheCvtExStrOutA(CACHE_EXSTRP src, CACHE_ASTRP tbl, CACHE_EXSTRP res)
```

Arguments

src	Address of a CACHE_EXSTRP variable that contains the string to be converted.
tbl	The name of the I/O translation table to use to perform the translation (a null string indicates that the default process I/O translation table name should be used).
res	Address of a CACHE_EXSTRP variable that will contain the result.

Description

Translates a string from the local 8-bit character string encoding used internally in the Caché 8-bit product to a string with the specified external character set encoding. (This is only available with 8-bit versions of Caché.)

Return Values for CacheCvtExStrOutA

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_ERRUNIMPLEMENTED	Not available for Unicode.
CACHE_ERVALUE	The specified I/O translation table name was undefined or did not have an input component.
CACHE_ERXLATE	Input string could not be translated using the specified I/O translation table.
CACHE_NOCON	No connection has been established.
CACHE_RETTRUNC	Result was truncated because result buffer was too small.
CACHE_FAILURE	Error encountered while trying to build translation data structures (probably not enough partition memory).
CACHE_SUCCESS	Translation completed successfully.

4.17 CacheCvtExStrOutW

Variants: **CacheCvtExStrOutA**, **CacheCvtExStrOutH**

```
int CacheCvtExStrOutW(CACHE_EXSTRP src, CACHEWSTRP tbl, CACHE_EXSTRP res)
```

Arguments

src	Address of a CACHE_EXSTRP variable that contains the string to be converted.
tbl	The name of the I/O translation table to use to perform the translation (a null string indicates that the default process I/O translation table name should be used).
res	Address of a CACHE_EXSTRP variable that will contain the result.

Description

Translates a string from the 2–byte Unicode character string encoding used internally in Unicode versions of Caché to a string with the specified external character set encoding. (This is only available with Unicode versions of Caché.)

Return Values for CacheCvtExStrOutW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_ERRUNIMPLEMENTED	Not available for 8–bit systems.
CACHE_ERVALUE	The specified I/O translation table name was undefined or did not have an input component.
CACHE_ERXLATE	Input string could not be translated using the specified I/O translation table.
CACHE_NOCON	No connection has been established.
CACHE_RETTRUNC	Result was truncated because result buffer was too small.
CACHE_FAILURE	Error encountered while trying to build translation data structures (probably not enough partition memory).
CACHE_SUCCESS	Translation completed successfully.

4.18 CacheCvtExStrOutH

Variants: **CacheCvtExStrOutA**, **CacheCvtExStrOutW**

```
int CacheCvtExStrOutH(CACHE_EXSTRP src, CACHEWSTRP tbl, CACHE_EXSTRP res)
```

Arguments

src	Address of a CACHE_EXSTRP variable that contains the string to be converted.
tbl	The name of the I/O translation table to use to perform the translation (a null string indicates that the default process I/O translation table name should be used).
res	Address of a CACHE_EXSTRP variable that will contain the result.

Description

Translates a string from the 4–byte Unicode character string encoding used internally in Unicode versions of Caché to a string with the specified external character set encoding. (This is only available with Unicode versions of Caché.)

Return Values for CacheCvtExStrOutH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_ERRUNIMPLEMENTED	Not available for 8–bit systems.
CACHE_ERVALUE	The specified I/O translation table name was undefined or did not have an input component.
CACHE_ERXLATE	Input string could not be translated using the specified I/O translation table.
CACHE_NOCON	No connection has been established.
CACHE_RETTRUNC	Result was truncated because result buffer was too small.
CACHE_FAILURE	Error encountered while trying to build translation data structures (probably not enough partition memory).
CACHE_SUCCESS	Translation completed successfully.

4.19 CacheCvtInA

Variants: **CacheCvtInW**, **CacheCvtInH**

```
int CacheCvtInA(CACHE_ASTRP src, CACHE_ASTRP tbl, CACHE_ASTRP res)
```

Arguments

src	The string in an external character set encoding to be translated (described using a counted character string buffer). The string should be initialized, for example, by setting the value to the number of blanks representing the maximum number of characters expected as output.
tbl	The name of the I/O translation table to use to perform the translation (a null string indicates that the default process I/O translation table name should be used).
res	Address of a CACHE_ASTR variable that will contain the counted 8-bit string result.

Description

Translates string with specified external character set encoding to the local 8-bit character string encoding used internally only in 8-bit versions of Caché.

Return Values for CacheCvtInA

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_ERRUNIMPLEMENTED	Not available for Unicode.
CACHE_ERVALUE	The specified I/O translation table name was undefined or did not have an input component.
CACHE_ERXLATE	Input string could not be translated using the specified I/O translation table.
CACHE_NOCON	No connection has been established.
CACHE_RETTRUNC	Result was truncated because result buffer was too small.
CACHE_FAILURE	Error encountered while trying to build translation data structures (probably not enough partition memory).
CACHE_SUCCESS	Translation completed successfully.

4.20 CacheCvtInH

Variants: **CacheCvtInA**, **CacheCvtInW**

```
int CacheCvtInH(CACHE_ASTRP src, CACHEHSTRP tbl, CACHEHSTRP res)
```

Arguments

src	The string in an external character set encoding to be translated (described using the number of bytes required to hold the Unicode string).
tbl	The name of the I/O translation table to use to perform the translation (a null string indicates that the default process I/O translation table name should be used).
res	Address of a CACHEHSTRP variable that will contain the counted Unicode string result.

Description

Translates string with specified external character set encoding to the Unicode character string encoding used internally in Unicode versions of Caché.

Return Values for CacheCvtInH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_ERRUNIMPLEMENTED	Not available for 8–bit systems.
CACHE_ERVALUE	The specified I/O translation table name was undefined or did not have an input component.
CACHE_ERXLATE	Input string could not be translated using the specified I/O translation table.
CACHE_NOCON	No connection has been established.
CACHE_RETTRUNC	Result was truncated because result buffer was too small.
CACHE_FAILURE	Error encountered while trying to build translation data structures (probably not enough partition memory).
CACHE_SUCCESS	Translation completed successfully.

4.21 CacheCvtInW

Variants: **CacheCvtInA**, **CacheCvtInH**

```
int CacheCvtInW(CACHE_ASTRP src, CACHEWSTRP tbl, CACHEWSTRP res)
```

Arguments

src	The string in an external character set encoding to be translated (described using the number of bytes required to hold the Unicode string).
tbl	The name of the I/O translation table to use to perform the translation (a null string indicates that the default process I/O translation table name should be used).
res	Address of a CACHEWSTR variable that will contain the counted Unicode string result.

Description

Translates string with specified external character set encoding to the Unicode character string encoding used internally in Unicode versions of Caché.

Return Values for CacheCvtInW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_ERRUNIMPLEMENTED	Not available for 8–bit systems.
CACHE_ERVALUE	The specified I/O translation table name was undefined or did not have an input component.
CACHE_ERXLATE	Input string could not be translated using the specified I/O translation table.
CACHE_NOCON	No connection has been established.
CACHE_RETTRUNC	Result was truncated because result buffer was too small.
CACHE_FAILURE	Error encountered while trying to build translation data structures (probably not enough partition memory).
CACHE_SUCCESS	Translation completed successfully.

4.22 CacheCvtOutA

Variants: **CacheCvtOutW**, **CacheCvtOutH**

```
int CacheCvtOutA(CACHE_ASTRP src, CACHE_ASTRP tbl, CACHE_ASTRP res)
```

Arguments

src	The string in the local 8-bit character string encoding used internally in the Caché 8-bit product (if a NULL pointer is passed, Caché will use the result from the last call to **CacheEvalA** or **CacheEvalW**).
tbl	The name of the I/O translation table to use to perform the translation (a null string indicates that the default process I/O translation table name should be used).
res	Address of a CACHE_ASTR variable that will contain the result in the target external character set encoding (described using a counted 8-bit character string buffer).

Description

Translates a string from the local 8-bit character string encoding used internally in the Caché 8-bit product to a string with the specified external character set encoding. (This is only available with 8-bit versions of Caché.)

Return Values for CacheCvtOutA

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_ERRUNIMPLEMENTED	Not available for Unicode.
CACHE_ERVALUE	The specified I/O translation table name was undefined or did not have an input component.
CACHE_ERXLATE	Input string could not be translated using the specified I/O translation table.
CACHE_NOCON	No connection has been established.
CACHE_RETTRUNC	Result was truncated because result buffer was too small.
CACHE_FAILURE	Error encountered while trying to build translation data structures (probably not enough partition memory).
CACHE_SUCCESS	Translation completed successfully.

4.23 CacheCvtOutH

Variants: **CacheCvtOutA**, **CacheCvtOutW**

```
int CacheCvtOutH(CACHEHSTRP src, CACHEHSTRP tbl, CACHE_ASTRP res)
```

Arguments

src	The string in the Unicode character string encoding used internally in the Caché Unicode product (if a NULL pointer is passed, Caché will use the result from the last call to **CacheEvalA** or **CacheEvalW**).
tbl	The name of the I/O translation table to use to perform the translation (a null string indicates that the default process I/O translation table name should be used).
res	Address of a CACHE_ASTR variable that will contain the result in the target external character set encoding (described using a counted 8-bit character string buffer).

Description

Translates a string from the Unicode character string encoding used internally in Unicode versions of Caché to a string with the specified external character set encoding. (This is only available with Unicode versions of Caché.)

Return Values for CacheCvtOutH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_ERRUNIMPLEMENTED	Not available for 8–bit systems.
CACHE_ERVALUE	The specified I/O translation table name was undefined or did not have an input component.
CACHE_ERXLATE	Input string could not be translated using the specified I/O translation table.
CACHE_NOCON	No connection has been established.
CACHE_RETTRUNC	Result was truncated because result buffer was too small.
CACHE_FAILURE	Error encountered while trying to build translation data structures (probably not enough partition memory).
CACHE_SUCCESS	Translation completed successfully.

4.24 CacheCvtOutW

Variants: **CacheCvtOutA**, **CacheCvtOutH**

```
int CacheCvtOutW(CACHEWSTRP src, CACHEWSTRP tbl, CACHE_ASTRP res)
```

Arguments

src	The string in the Unicode character string encoding used internally in the Caché Unicode product (if a NULL pointer is passed, Caché will use the result from the last call to **CacheEvalA** or **CacheEvalW**).
tbl	The name of the I/O translation table to use to perform the translation (a null string indicates that the default process I/O translation table name should be used).
res	Address of a CACHE_ASTR variable that will contain the result in the target external character set encoding (described using a counted 8-bit character string buffer).

Description

Translates a string from the Unicode character string encoding used internally in Unicode versions of Caché to a string with the specified external character set encoding. (This is only available with Unicode versions of Caché.)

Return Values for CacheCvtOutW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_ERRUNIMPLEMENTED	Not available for 8–bit systems.
CACHE_ERVALUE	The specified I/O translation table name was undefined or did not have an input component.
CACHE_ERXLATE	Input string could not be translated using the specified I/O translation table.
CACHE_NOCON	No connection has been established.
CACHE_RETTRUNC	Result was truncated because result buffer was too small.
CACHE_FAILURE	Error encountered while trying to build translation data structures (probably not enough partition memory).
CACHE_SUCCESS	Translation completed successfully.

4.25 CacheDoFun

```
int CacheDoFun(unsigned int flags, int narg)
```

Arguments

flags	Routine flags from **CachePushRtn[XW]**
narg	Number of call arguments pushed onto the argument stack. Target must have a (possibly empty) formal parameter list.

Description

Performs a routine call (special case).

Return Values for CacheDoFun

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_FAILURE	Internal consistency error.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.26 CacheDoRtn

```
int CacheDoRtn(unsigned int flags, int narg)
```

Arguments

flags	Routine flags from **CachePushRtn[XW]**
narg	Number of call arguments pushed onto the argument stack. If zero, target must not have a formal parameter list.

Description

Performs a routine call.

Return Values for CacheDoRtn

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_FAILURE	Internal consistency error.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.27 CacheEnd

```
int CacheEnd()
```

Description

Terminates a Caché process. If there is a broken connection, it also performs clean-up operations.

Return Values for CacheEnd

CACHE_FAILURE	Returns if called from a **$ZF** function (rather than from within a Callin executable).
CACHE_NOCON	No connection has been established.
CACHE_SUCCESS	Caché session terminated/cleaned up.

CacheEnd can also return any of the Caché error codes.

Example

```
rc = CacheEnd();
```

4.28 CacheEndAll

```
int CacheEndAll()
```

Description

Disconnects all threads in a threaded Callin environment, then schedules the threads for termination and waits until they are done.

Return Values for CacheEndAll

CACHE_SUCCESS	Caché session terminated/cleaned up.

Example

```
rc = CacheEndAll();
```

4.29 CacheErrorA

Variants: **CacheErrorW**, **CacheErrorH**

```
int CacheErrorA(CACHE_ASTRP msg, CACHE_ASTRP src, int * offp)
```

Arguments

msg	The error message or the address of a variable to receive the error message.
src	The source string for the error or the address of a variable to receive the source string the error message.
offp	An integer that specifies the offset to location in *errsrc* or the address of an integer to receive the offset to the source string the error message.

Description

Returns the most recent error message, its associated source string, and the offset to where in the source string the error occurred.

Return Values for CacheErrorA

CACHE_CONBROKEN	Connection has been broken.
CACHE_NOCON	No connection has been established.
CACHE_RETTOOSMALL	The length of the return value for either *errmsg* or *errsrc* was not of the valid size.
CACHE_SUCCESS	Connection formed.

Example

```
CACHE_ASTR errmsg;
CACHE_ASTR srcline;
int offset;
errmsg.len = 50;
srcline.len = 100;
if ((rc = CacheErrorA(&errmsg, &srcline, &offset)) != CACHE_SUCCESS)
printf("\r\nfailed to display error - rc = %d",rc);
```

4.30 CacheErrorH

Variants: **CacheErrorA**, **CacheErrorW**

```
int CacheErrorH(CACHEHSTRP msg, CACHEHSTRP src, int * offp)
```

Arguments

msg	The error message or the address of a variable to receive the error message.
src	The source string for the error or the address of a variable to receive the source string the error message.
offp	The offset to location in *errsrc* or the address of an integer to receive the offset to the source string the error message.

Description

Returns the most recent error message, its associated source string, and the offset to where in the source string the error occurred.

Return Values for CacheErrorH

CACHE_CONBROKEN	Connection has been broken.
CACHE_NOCON	No connection has been established.
CACHE_RETTOOSMALL	The length of the return value for either *errmsg* or *errsrc* was not of the valid size.
CACHE_SUCCESS	Connection formed.

Example

```
CACHEHSTRP errmsg;
CACHEHSTRP srcline;
int offset;
errmsg.len = 50;
srcline.len = 100;
if ((rc = CacheErrorH(&errmsg, &srcline, &offset)) != CACHE_SUCCESS)
printf("\r\nfailed to display error - rc = %d",rc);
```

4.31 CacheErrorW

Variants: **CacheErrorA**, **CacheErrorH**

```
int CacheErrorW(CACHEWSTRP msg, CACHEWSTRP src, int * offp)
```

Arguments

msg	The error message or the address of a variable to receive the error message.
src	The source string for the error or the address of a variable to receive the source string the error message.
offp	The offset to location in *errsrc* or the address of an integer to receive the offset to the source string the error message.

Description

Returns the most recent error message, its associated source string, and the offset to where in the source string the error occurred.

Return Values for CacheErrorW

CACHE_CONBROKEN	Connection has been broken.
CACHE_NOCON	No connection has been established.
CACHE_RETTOOSMALL	The length of the return value for either *errmsg* or *errsrc* was not of the valid size.
CACHE_SUCCESS	Connection formed.

Example

```
CACHEWSTRP errmsg;
CACHEWSTRP srcline;
int offset;
errmsg.len = 50;
srcline.len = 100;
if ((rc = CacheErrorW(&errmsg, &srcline, &offset)) != CACHE_SUCCESS)
printf("\r\nfailed to display error - rc = %d",rc);
```

4.32 CacheErrxlateA

Variants: **CacheErrxlateW**, **CacheErrxlateH**

```
int CacheErrxlateA(int code, CACHE_ASTRP rbuf)
```

Arguments

code	The error code.
rbuf	Address of a CACHE_ASTR variable to contain the Caché error string. The *len* field should be loaded with the maximum string size that can be returned.

Description

Translates error code *code* into a Cache error string, and writes that string into the structure pointed to by *rbuf*

Return Values for CacheErrxlateA

CACHE_ERUNKNOWN	The specified code is less than 1 (in the range of the Callin interface errors) or is above the largest Caché error number.
CACHE_RETTRUNC	The associated error string was truncated to fit in the allocated area.
CACHE_SUCCESS	Connection formed.

Example

```
CACHE_ASTR retval; /* define variable retval */
retval.len = 30; /* maximum return length of string */
rc = CacheErrxlateA(CACHE_ERSTORE,&retval);
```

4.33 CacheErrxlateH

Variants: **CacheErrxlateA**, **CacheErrxlateW**

```
int CacheErrxlateH(int code, CACHEHSTRP rbuf)
```

Arguments

code	The error code.
rbuf	Address of a CACHEHSTRP variable to contain the Caché error string. The *len* field should be loaded with the maximum string size that can be returned.

Description

Translates error code *code* into a Cache error string, and writes that string into the structure pointed to by *rbuf*

Return Values for CacheErrxlateH

CACHE_ERUNKNOWN	The specified code is less than 1 (in the range of the Callin interface errors) or is above the largest Caché error number.
CACHE_RETTRUNC	The associated error string was truncated to fit in the allocated area.
CACHE_SUCCESS	Connection formed.

Example

```
CACHEHSTR retval; /* define variable retval */
retval.len = 30; /* maximum return length of string */
rc = CacheErrxlateH(CACHE_ERSTORE,&retval);
```

4.34 CacheErrxlateW

Variants: **CacheErrxlateA**, **CacheErrxlateH**

```
int CacheErrxlateW(int code, CACHEWSTRP rbuf)
```

Arguments

code	The error code.
rbuf	Address of a CACHEWSTR variable to contain the Caché error string. The *len* field should be loaded with the maximum string size that can be returned.

Description

Translates error code *code* into a Cache error string, and writes that string into the structure pointed to by *rbuf*

Return Values for CacheErrxlateW

CACHE_ERUNKNOWN	The specified code is less than 1 (in the range of the Callin interface errors) or is above the largest Caché error number.
CACHE_RETTRUNC	The associated error string was truncated to fit in the allocated area.
CACHE_SUCCESS	Connection formed.

Example

```
CACHEWSTR retval; /* define variable retval */
retval.len = 30; /* maximum return length of string */
rc = CacheErrxlateW(CACHE_ERSTORE,&retval);
```

4.35 CacheEvalA

Variants: **CacheEvalW**, **CacheEvalH**

```
int CacheEvalA(CACHE_ASTRP volatile expr)
```

Arguments

expr	The address of a CACHE_ASTR variable.

Description

Evaluates a string as if it were a Caché expression and places the return value in memory for further processing by **CacheType** and **CacheConvert**.

If **CacheEvalA** completes successfully, it sets a flag that allows calls to **CacheType** and **CacheConvert** to complete. These functions are used to process the item returned from **CacheEvalA**.

CAUTION: The next call to **CacheEvalA**, **CacheExecuteA**, or **CacheEnd** will overwrite the existing return value.

Return Values for CacheEvalA

CACHE_CONBROKEN	Connection has been closed due to a serious error condition or **RESJOB**.
CACHE_ERSYSTEM	Either Caché generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
CACHE_NOCON	No connection has been established.
CACHE_STRTOOLONG	String is too long.
CACHE_SUCCESS	String evaluated successfully.

CacheEvalA can also return any of the Caché error codes.

Example

```
int rc;
CACHE_ASTR retval;
CACHE_ASTR expr;

strcpy(expr.str, "\"Record\"_^Recnum_\" = \"_$$^GetRec(^Recnum)");
expr.len = strlen(expr.str);
rc = CacheEvalA(&expr);
if (rc == CACHE_SUCCESS)
    rc = CacheConvert(CACHE_ASTRING,&retval);
```

4.36 CacheEvalH

Variants: **CacheEvalA**, **CacheEvalW**

```
int CacheEvalH(CACHEHSTRP volatile expr)
```

Arguments

expr	The address of a CACHEHSTRP variable.

Description

Evaluates a string as if it were a Caché expression and places the return value in memory for further processing by **CacheType** and **CacheConvert**.

If **CacheEvalH** completes successfully, it sets a flag that allows calls to **CacheType** and **CacheConvert** to complete. These functions are used to process the item returned from **CacheEvalA**.

CAUTION: The next call to **CacheEvalH**, **CacheExecuteH**, or **CacheEnd** will overwrite the existing return value.

Return Values for CacheEvalH

CACHE_CONBROKEN	Connection has been closed due to a serious error condition or **RESJOB**.
CACHW_ERSYSTEM	Either Caché generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
CACHE_NOCON	No connection has been established.
CACHE_STRTOOLONG	String is too long.
CACHE_SUCCESS	String evaluated successfully.

CacheEvalH can also return any of the Caché error codes.

Example

```
int rc;
CACHEHSTRP retval;
CACHEHSTRP expr;

strcpy(expr.str, "\"Record\"_^Recnum_\" = \"_$$^GetRec(^Recnum)");
expr.len = strlen(expr.str);
rc = CacheEvalH(&expr);
if (rc == CACHE_SUCCESS)
    rc = CacheConvert(ING,&retval);
```

4.37 CacheEvalW

Variants: **CacheEvalA**, **CacheEvalH**

```
int CacheEvalW(CACHEWSTRP volatile expr)
```

Arguments

expr	The address of a CACHEWSTR variable.

Description

Evaluates a string as if it were a Caché expression and places the return value in memory for further processing by **CacheType** and **CacheConvert**.

If **CacheEvalW** completes successfully, it sets a flag that allows calls to **CacheType** and **CacheConvert** to complete. These functions are used to process the item returned from **CacheEvalA**.

CAUTION: The next call to **CacheEvalW**, **CacheExecuteW**, or **CacheEnd** will overwrite the existing return value.

Return Values for CacheEvalW

CACHE_CONBROKEN	Connection has been closed due to a serious error condition or **RESJOB**.
CACHW_ERSYSTEM	Either Caché generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
CACHE_NOCON	No connection has been established.
CACHE_STRTOOLONG	String is too long.
CACHE_SUCCESS	String evaluated successfully.

CacheEvalW can also return any of the Caché error codes.

Example

```
int rc;
CACHEWSTR retval;
CACHEWSTR expr;

strcpy(expr.str, "\"Record\"_^Recnum_\" = \"_$$^GetRec(^Recnum)");
expr.len = strlen(expr.str);
rc = CacheEvalW(&expr);
if (rc == CACHE_SUCCESS)
    rc = CacheConvert(ING,&retval);
```

4.38 CacheExecuteA

Variants: **CacheExecuteW**, **CacheExecuteH**

```
int CacheExecuteA(CACHE_ASTRP volatile cmd)
```

Arguments

cmd	The address of a CACHE_ASTR variable.

Description

Executes the command *string* as if it were typed at the Caché programmer prompt.

CAUTION: The next call to **CacheEvalA**, **CacheExecuteA**, or **CacheEnd** will overwrite the existing return value.

Return Values for CacheExecuteA

CACHE_CONBROKEN	Connection has been closed due to a serious error condition or **RESJOB**.
CACHE_ERSYSTEM	Either ObjectScript generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
CACHE_NOCON	No connection has been established.
CACHE_STRTOOLONG	String is too long.
CACHE_SUCCESS	String executed successfully.

CacheExecuteA can also return any of the Caché error codes.

Example

```
int rc;
CACHE_ASTR command;
sprintf(command.str,"ZN \"USER\""); /* changes namespace */
command.len = strlen(command.str);
rc = CacheExecuteA(&command);
```

4.39 CacheExecuteH

Variants: **CacheExecuteA**, **CacheExecuteW**

```
int CacheExecuteH(CACHEHSTRP volatile cmd)
```

Arguments

cmd	The address of a CACHE_ASTR variable.

Description

Executes the command *string* as if it were typed at the Caché programmer prompt.

If **CacheExecuteH** completes successfully, it sets a flag that allows calls to **CacheType** and **CacheConvert** to complete. These functions are used to process the item returned from **CacheEvalH**.

CAUTION: The next call to **CacheEvalH**, **CacheExecuteH**, or **CacheEnd** will overwrite the existing return value.

Return Values for CacheExecuteH

CACHE_CONBROKEN	Connection has been closed due to a serious error condition or **RESJOB**.
CACHE_ERSYSTEM	Either ObjectScript generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
CACHE_NOCON	No connection has been established.
CACHE_STRTOOLONG	String is too long.
CACHE_SUCCESS	String executed successfully.

CacheExecuteH can also return any of the Caché error codes.

Example

```
int rc;
unsigned short zname[] = {'Z','N',' ','"','U','S','E','R','"'};
CACHEHSTRP pcommand;
pcommand.str = zname;
pcommand.len = sizeof(zname) / sizeof(unsigned short);
rc = CacheExecuteH(pcommand);
```

4.40 CacheExecuteW

Variants: **CacheExecuteA**, **CacheExecuteH**

```
int CacheExecuteW(CACHEWSTRP volatile cmd)
```

Arguments

cmd	The address of a CACHE_ASTR variable.

Description

Executes the command *string* as if it were typed at the Caché programmer prompt.

If **CacheExecuteW** completes successfully, it sets a flag that allows calls to **CacheType** and **CacheConvert** to complete. These functions are used to process the item returned from **CacheEvalW**.

CAUTION: The next call to **CacheEvalW**, **CacheExecuteW**, or **CacheEnd** will overwrite the existing return value.

Return Values for CacheExecuteW

CACHE_CONBROKEN	Connection has been closed due to a serious error condition or **RESJOB**.
CACHE_ERSYSTEM	Either ObjectScript generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
CACHE_NOCON	No connection has been established.
CACHE_STRTOOLONG	String is too long.
CACHE_SUCCESS	String executed successfully.

CacheExecuteW can also return any of the Caché error codes.

Example

```
int rc;
unsigned short zname[] = {'Z','N',' ','"','U','S','E','R','"'};
CACHEWSTRP pcommand;
pcommand.str = zname;
pcommand.len = sizeof(zname) / sizeof(unsigned short);
rc = CacheExecuteW(pcommand);
```

4.41 CacheExStrKill

```
int CacheExStrKill(CACHE_EXSTRP obj)
```

Arguments

obj	Pointer to the string.

Description

Releases the storage associated with an EXSTR string.

Return Values for CacheExStrKill

CACHE_ERUNIMPLEMENTED	String is undefined.
CACHE_SUCCESS	String storage has been released.

4.42 CacheExStrNew

Variants: **CacheExStrNewW**, **CacheExStrNewH**

```
unsigned char * CacheExStrNew(CACHE_EXSTRP zstr, int size)
```

Arguments

zstr	Pointer to a CACHE_EXSTR string descriptor.
size	Number of 8–bit characters to allocate.

Description

Allocates the requested amount of storage for a string, and fills in the EXSTR structure with the length and a pointer to the value field of the structure.

Return Values for CacheExStrNew

Returns a pointer to the allocated string, or NULL if no string was allocated.

4.43 CacheExStrNewW

Variants: **CacheExStrNew**, **CacheExStrNewH**

```
unsigned short * CacheExStrNewW(CACHE_EXSTRP zstr, int size)
```

Arguments

zstr	Pointer to a CACHE_EXSTR string descriptor.
size	Number of 2–byte characters to allocate.

Description

Allocates the requested amount of storage for a string, and fills in the EXSTR structure with the length and a pointer to the value field of the structure.

Return Values for CacheExStrNewW

Returns a pointer to the allocated string, or NULL if no string was allocated.

4.44 CacheExStrNewH

Variants: **CacheExStrNew**, **CacheExStrNewW**

```
unsigned short * CacheExStrNewH(CACHE_EXSTRP zstr, int size)
```

Arguments

zstr	Pointer to a CACHE_EXSTR string descriptor.
size	Number of 4–byte characters to allocate.

Description

Allocates the requested amount of storage for a string, and fills in the EXSTR structure with the length and a pointer to the value field of the structure.

Return Values for CacheExStrNewH

Returns a pointer to the allocated string, or NULL if no string was allocated.

4.45 CacheExtFun

```
int CacheExtFun(unsigned int flags, int narg)
```

Arguments

flags	Routine flags from **CachePushFunc[XW]**.
narg	Number of call arguments pushed onto the argument stack.

Description

Performs an extrinsic function call where the return value is pushed onto the argument stack.

Return Values for CacheExtFun

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_FAILURE	Internal consistency error.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.46 CacheGetProperty

```
int CacheGetProperty()
```

Description

Obtains the value of the property defined by **CachePushProperty**. The value is pushed onto the argument stack.

Return Values for CacheGetProperty

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.47 CacheGlobalData

```
int CacheGlobalData(int narg, int valueflag)
```

Arguments

narg	Number of call arguments pushed onto the argument stack.
valueflag	Indicates whether the data value, if there is one, should be returned.

Description

Performs a $Data on the specified global.

Return Values for CacheGlobalData

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_ERPROTECT	Protection violation.
CACHE_ERUNDEF	Node has no associated value.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.48 CacheGlobalGet

```
int CacheGlobalGet(int narg, int flag)
```

Arguments

narg	Number of subscript expressions pushed onto the argument stack.
flag	Indicates behavior when global reference is undefined: • `0` — returns CACHE_ERUNDEF • `1` — returns CACHE_SUCCESS but the return value is an empty string.

Description

Obtains the value of the global reference defined by **CachePushGlobal** and any subscripts. The node value is pushed onto the argument stack.

Return Values for CacheGlobalGet

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_ERPROTECT	Protection violation.
CACHE_ERUNDEF	Node has no associated value.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.49 CacheGlobalGetBinary

```
int CacheGlobalGetBinary(int numsub, int flag, int *plen, Callin_char_t **pbuf)
```

Arguments

numsub	Number of subscript expressions pushed onto the argument stack.
flag	Indicates behavior when global reference is undefined: • `0` — returns CACHE_ERUNDEF • `1` — returns CACHE_SUCCESS but the return value is an empty string.
plen	Pointer to length of buffer.
pbuf	Pointer to buffer pointer.

Description

Obtains the value of the global reference defined by **CachePushGlobal**[**W**][**H**] and any subscripts, and also tests to make sure that the result is a binary string that will fit in the provided buffer. The node value is pushed onto the argument stack.

Return Values for CacheGlobalGetBinary

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_ERPROTECT	Protection violation.
CACHE_ERUNDEF	Node has no associated value.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.50 CacheGlobalIncrement

```
int CacheGlobalIncrement(int narg)
```

Arguments

narg	Number of call arguments pushed onto the argument stack.

Description

Performs a $INCREMENT and returns the result on top of the stack.

Return Values for CacheGlobalIncrement

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_ERPROTECT	Protection violation.
CACHE_ERUNDEF	Node has no associated value.
CACHE_ERMAXINCR	MAXINCREMENT system error
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.51 CacheGlobalKill

```
int CacheGlobalKill(int narg, int nodeonly)
```

Arguments

narg	Number of call arguments pushed onto the argument stack.
nodeonly	A value of 1 indicates that only the specified node should be killed. When the value is 0, the entire specified global tree is killed.

Description

Performs a ZKILL on a global node or tree.

Return Values for CacheGlobalKill

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_ERPROTECT	Protection violation.
CACHE_ERUNDEF	Node has no associated value.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.52 CacheGlobalOrder

```
int CacheGlobalOrder(int narg, int dir, int valueflag)
```

Arguments

narg	Number of call arguments pushed onto the argument stack.
dir	Direction for the $Order is 1 for forward, -1 for reverse.
valueflag	Indicates whether the data value, if there is one, should be returned.

Description

Performs a $Order on the specified global.

Return Values for CacheGlobalOrder

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_ERPROTECT	Protection violation.
CACHE_ERUNDEF	Node has no associated value.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.53 CacheGlobalQuery

```
int CacheGlobalQuery(int narg, int dir, int valueflag)
```

Arguments

narg	Number of call arguments pushed onto the argument stack.
dir	Direction for the $Query is 1 for forward, -1 for reverse.
valueflag	Indicates whether the data value, if there is one, should be returned.

Description

Performs a $Query on the specified global.

Return Values for CacheGlobalQuery

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_ERPROTECT	Protection violation.
CACHE_ERUNDEF	Node has no associated value.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.54 CacheGlobalRelease

```
int CacheGlobalRelease( )
```

Description

Release ownership of a retained global buffer, if one exists.

Return Values for CacheGlobalRelease

CACHE_SUCCESS	The operation was successful.

4.55 CacheGlobalSet

```
int CacheGlobalSet(int narg)
```

Arguments

narg	Number of subscript expressions pushed onto the argument stack.

Description

Stores the value of the global reference defined by **CachePushGlobal** and any subscripts. The node value must be pushed onto the argument stack before this call.

Return Values for CacheGlobalSet

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.56 CacheIncrementCountOref

```
int CacheIncrementCountOref(unsigned int oref)
```

Arguments

oref	Object reference.

Description

Increments the system reference counter for an OREF.

Return Values for CacheIncrementCountOref

CACHE_ERBADOREF	Invalid OREF.
CACHE_SUCCESS	The operation was successful.

4.57 CacheInvokeClassMethod

```
int CacheInvokeClassMethod(int narg)
```

Arguments

narg	Number of call arguments pushed onto the argument stack.

Description

Executes the class method call defined by **CachePushClassMethod[W]** and any arguments. The return value is pushed onto the argument stack.

Return Values for CacheInvokeClassMethod

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.58 CacheInvokeMethod

```
int CacheInvokeMethod(int narg)
```

Arguments

narg	Number of call arguments pushed onto the argument stack.

Description

Executes the instance method call defined by **CachePushMethod[W]** and any arguments pushed onto the argument stack.

Return Values for CacheInvokeMethod

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.59 CacheOflush

```
int CacheOflush()
```

Description

Flushes any pending output.

Return Values for CacheOflush

CACHE_FAILURE	Returns if called from a **$ZF** function (rather than from within a Callin executable).
CACHE_SUCCESS	Control function performed.

4.60 CachePop

```
int CachePop(void ** arg)
```

Arguments

arg	Pointer to argument stack entry.

Description

Pops a value off argument stack.

Return Values for CachePop

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_SUCCESS	The operation was successful.

4.61 CachePopCvtH

Variants: **CachePopCvtW**

```
int CachePopCvtH(int * lenp, wchar_t ** strp)
```

Arguments

lenp	Pointer to length of string.
strp	Pointer to string pointer.

Description

Pops a local 8-bit string off argument stack and translates it to 4–byte Unicode. Identical to **CachePopStrH** in Unicode environments.

Return Values for CachePopCvtH

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.62 CachePopCvtW

Variants: **CachePopCvtH**

```
int CachePopCvtW(int * lenp, unsigned short ** strp)
```

Arguments

lenp	Pointer to length of string.
strp	Pointer to string pointer.

Description

Deprecated: The long string function **CachePopExStrCvtW** should be used for all strings.

Pops a local 8-bit string off argument stack and translates it to 2–byte Unicode. Identical to **CachePopStrW** in Unicode environments.

Return Values for CachePopCvtW

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.63 CachePopDbl

```
int CachePopDbl(double * nump)
```

Arguments

nump	Pointer to double value.

Description

Pops a value off argument stack and converts it to a double.

Return Values for CachePopDbl

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_SUCCESS	The operation was successful.

4.64 CachePopExStr

Variants: **CachePopExStrW**, **CachePopExStrH**

```
int CachePopExStr(CACHE_EXSTRP sstrp)
```

Arguments

sstrp	Pointer to long string pointer.

Description

Pops a value off argument stack and converts it to a string in local 8–bit encoding.

Return Values for CachePopExStr

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_SUCCESS	The operation was successful.
CACHE_EXSTR_INUSE	Returned if *sstrp* has not been initialized to `NULL`.

4.65 CachePopExStrCvtW

Variants: **CachePopExStrCvtH**

```
int CachePopExStrCvtW(CACHE_EXSTRP sstr)
```

Arguments

sstr	Pointer to long string pointer.

Description

Pops a local 8-bit string off the argument stack and translates it to a 2–byte Unicode string. On Unicode systems, this is the same as CachePopExStrW.

Return Values for CachePopExStrCvtW

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.66 CachePopExStrCvtH

Variants: **CachePopExStrCvtW**

```
int CachePopExStrCvtW(CACHE_EXSTRP sstr)
```

Arguments

sstr	Pointer to long string pointer.

Description

Pops a local 8-bit string off argument stack and translates it to a 4–byte Unicode string. On Unicode systems, this is the same as CachePopExStrH.

Return Values for CachePopExStrCvtH

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.67 CachePopExStrW

Variants: **CachePopExStr**, **CachePopExStrH**

```
int CachePopExStrW(CACHE_EXSTRP sstrp)
```

Arguments

sstrp	Pointer to long string pointer.

Description

Pops a value off argument stack and converts it to a 2–byte Unicode string.

Return Values for CachePopExStrW

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
CACHE_EXSTR_INUSE	Returned if *sstrp* has not been initialized to `NULL`.

4.68 CachePopExStrH

Variants: **CachePopExStr**, **CachePopExStrW**

```
int CachePopExStrH(CACHE_EXSTRP sstrp)
```

Arguments

sstrp	Pointer to long string pointer.

Description

Pops a value off argument stack and converts it to a 4–byte Unicode string.

Return Values for CachePopExStrH

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
CACHE_EXSTR_INUSE	Returned if *sstrp* has not been initialized to `NULL`.

4.69 CachePopInt

```
int CachePopInt(int* nump)
```

Arguments

nump	Pointer to integer value.

Description

Pops a value off argument stack and converts it to an integer.

Return Values for CachePopInt

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_SUCCESS	The operation was successful.

4.70 CachePopInt64

```
int CachePopInt64(long long * nump)
```

Arguments

nump	Pointer to long long value.

Description

Pops a value off argument stack and converts it to a 64–bit (long long) value.

Return Values for CachePopInt64

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_SUCCESS	The operation was successful.

4.71 CachePopList

```
int CachePopList(int * lenp, Callin_char_t ** strp)
```

Arguments

lenp	Pointer to length of string.
strp	Pointer to string pointer.

Description

Pops a $LIST object off argument stack and converts it. String elements are copied or translated as appropriate depending on whether this is a Unicode or 8-bit version.

Return Values for CachePopList

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.72 CachePopOref

```
int CachePopOref(unsigned int * orefp)
```

Arguments

orefp	Pointer to OREF value.

Description

Pops an OREF off argument stack.

Return Values for CachePopOref

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_ERNOOREF	Result is not an OREF.
CACHE_SUCCESS	The operation was successful.

4.73 CachePopPtr

```
int CachePopPtr(void ** ptrp)
```

Arguments

ptrp	Pointer to generic pointer.

Description

Pops a pointer off argument stack in internal format.

Return Values for CachePopPtr

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_BADARG	The entry is not a valid pointer.
CACHE_SUCCESS	The operation was successful.

4.74 CachePopStr

Variants: **CachePopStrW**, **CachePopStrH**

```
int CachePopStr(int * lenp, Callin_char_t ** strp)
```

Arguments

lenp	Pointer to length of string.
strp	Pointer to string pointer.

Description

Pops a value off argument stack and converts it to a string.

Return Values for CachePopStr

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_SUCCESS	The operation was successful.

4.75 CachePopStrH

Variants: **CachePopStr**, **CachePopStrW**

```
int CachePopStrH(int * lenp, wchar_t ** strp)
```

Arguments

lenp	Pointer to length of string.
strp	Pointer to string pointer.

Description

Pops a value off argument stack and converts it to a 4-byte Unicode string.

Return Values for CachePopStrH

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.76 CachePopStrW

Variants: **CachePopStr**, **CachePopStrH**

```
int CachePopStrW(int * lenp, unsigned short ** strp)
```

Arguments

lenp	Pointer to length of string.
strp	Pointer to string pointer.

Description

Pops a value off argument stack and converts it to a 2-byte Unicode string.

Return Values for CachePopStrW

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.77 CachePromptA

Variants: **CachePromptW**, **CachePromptH**

```
int CachePromptA(CACHE_ASTRP rbuf)
```

Arguments

rbuf	The prompt string. The minimum length of the returned string is five characters.

Description

Returns a string that would be the programmer prompt (without the ">").

Return Values for CachePromptA

CACHE_CONBROKEN	Connection has been broken.
CACHE_ERSYSTEM	Either ObjectScript generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
CACHE_FAILURE	An unexpected error has occurred.
CACHE_NOCON	No connection has been established.
CACHE_RETTOOSMALL	*rbuf* must have a length of at least five.
CACHE_SUCCESS	Connection formed.

Example

```
CACHE_ASTR retval;      /* define variable retval */
retval.len = 5;         /* maximum return length of string */
rc = CachePromptA(&retval);
```

4.78 CachePromptH

Variants: **CachePromptA**, **CachePromptW**

```
int CachePromptH(CACHEHSTRP rbuf)
```

Arguments

rbuf	The prompt string. The minimum length of the returned string is five characters.

Description

Returns a string that would be the programmer prompt (without the ">").

Return Values for CachePromptH

CACHE_CONBROKEN	Connection has been broken.
CACHE_ERSYSTEM	Either ObjectScript generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
CACHE_FAILURE	Request failed.
CACHE_NOCON	No connection has been established.
CACHE_RETTOOSMALL	*rbuf* must have a length of at least five.
CACHE_SUCCESS	Connection formed.

Example

```
CACHEHSTRP retval; /* define variable retval */
retval.len = 5; /* maximum return length of string */
rc = CachePromptH( &retval);
```

4.79 CachePromptW

Variants: **CachePromptA**, **CachePromptH**

```
int CachePromptW(CACHEWSTRP rbuf)
```

Arguments

rbuf	The prompt string. The minimum length of the returned string is five characters.

Description

Returns a string that would be the programmer prompt (without the ">").

Return Values for CachePromptW

CACHE_CONBROKEN	Connection has been broken.
CACHE_ERSYSTEM	Either ObjectScript generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
CACHE_FAILURE	Request failed.
CACHE_NOCON	No connection has been established.
CACHE_RETTOOSMALL	*rbuf* must have a length of at least five.
CACHE_SUCCESS	Connection formed.

Example

```
CACHEWSTR retval; /* define variable retval */
retval.len = 5; /* maximum return length of string */
rc = CacheConvertW( &retval);
```

4.80 CachePushClassMethod

Variants: **CachePushClassMethodW**, **CachePushClassMethodH**

```
int CachePushClassMethod(int clen, const Callin_char_t * cptr,
                         int mlen, const Callin_char_t * mptr, int flg)
```

Arguments

clen	Class name length (characters).
cptr	Pointer to class name.
mlen	Method name length (characters).
mptr	Pointer to method name.
flg	Specifies whether the method will return a value. If the method returns a value, this flag must be set to `1` in order to retrieve it. The method must return a value via **Quit** with an argument. Set this parameter to `0` if no value will be returned.

Description

Pushes a class method reference onto the argument stack.

Return Values for CachePushClassMethod

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_BADARG	Invalid call argument.
CACHE_SUCCESS	The operation was successful.

4.81 CachePushClassMethodH

Variants: **CachePushClassMethod**, **CachePushClassMethodW**

```
int CachePushClassMethodH(int clen, const wchar_t * cptr,
                          int mlen, const wchar_t * mptr, int flg)
```

Arguments

clen	Class name length (characters).
cptr	Pointer to class name.
mlen	Method name length (characters).
mptr	Pointer to method name.
flg	Specifies whether the method will return a value. If the method returns a value, this flag must be set to `1` in order to retrieve it. The method must return a value via **Quit** with an argument. Set this parameter to `0` if no value will be returned.

Description

Pushes a 4-byte Unicode class method reference onto the argument stack.

Return Values for CachePushClassMethodH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_BADARG	Invalid call argument.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.82 CachePushClassMethodW

Variants: **CachePushClassMethod**, **CachePushClassMethodH**

```
int CachePushClassMethodW(int clen, const unsigned short * cptr,
                          int mlen, const unsigned short * mptr, int flg)
```

Arguments

clen	Class name length (characters).
cptr	Pointer to class name.
mlen	Method name length (characters).
mptr	Pointer to method name.
flg	Specifies whether the method will return a value. If the method returns a value, this flag must be set to `1` in order to retrieve it. The method must return a value via **Quit** with an argument. Set this parameter to `0` if no value will be returned.

Description

Pushes a 2-byte Unicode class method reference onto the argument stack.

Return Values for CachePushClassMethodW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_BADARG	Invalid call argument.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.83 CachePushCvtH

Variants: **CachePushCvtW**

```
int CachePushCvtH(int len, const wchar_t * ptr)
```

Arguments

len	Number of characters in string.
ptr	Pointer to string.

Description

Translates a Unicode string to local 8-bit and pushes it onto the argument stack. Identical to **CachePushStrH** for Unicode versions.

Return Values for CachePushCvtH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating the string.

4.84 CachePushCvtW

Variants: **CachePushCvtH**

```
int CachePushCvtW(int len, const unsigned short * ptr)
```

Arguments

len	Number of characters in string.
ptr	Pointer to string.

Description

Deprecated: The long string function **CachePushExStrCvtW** should be used for all strings.

Translates a Unicode string to local 8-bit and pushes it onto the argument stack. Identical to **CachePushStrW** for Unicode versions.

Return Values for CachePushCvtW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating the string.

4.85 CachePushDbl

```
int CachePushDbl(double num)
```

Arguments

num	Double value.

Description

Pushes a Caché double onto the argument stack.

Return Values for CachePushDbl

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_SUCCESS	The operation was successful.

4.86 CachePushExStr

Variants: **CachePushExStrW**, **CachePushExStrH**

```
int CachePushExStr(CACHE_EXSTRP sptr)
```

Arguments

sptr	Pointer to the argument value.

Description

Pushes a string onto the argument stack.

Return Values for CachePushExStr

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.87 CachePushExStrCvtW

Variants: **CachePushExStrCvtH**

```
int CachePushExStrCvtW(CACHE_EXSTRP sptr)
```

Arguments

sptr	Pointer to the argument value.

Description

Translates a Unicode string to local 8-bit and pushes it onto the argument stack.

Return Values for CachePushExStrCvtW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating the string.

4.88 CachePushExStrCvtH

Variants: **CachePushExStrCvtW**

```
int CachePushExStrCvtH(CACHE_EXSTRP sptr)
```

Arguments

sptr	Pointer to the argument value.

Description

Translates a 4–byte Unicode string to local 8-bit and pushes it onto the argument stack.

Return Values for CachePushExStrCvtH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating the string.

4.89 CachePushExStrW

Variants: **CachePushExStr**, **CachePushExStrH**

```
int CachePushExStrW(CACHE_EXSTRP sptr)
```

Arguments

sptr	Pointer to the argument value.

Description

Pushes a long Unicode string onto the argument stack.

Return Values for CachePushExStrW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.90 CachePushExStrH

Variants: **CachePushExStr**, **CachePushExStrW**

```
int CachePushExStrH(CACHE_EXSTRP sptr)
```

Arguments

sptr	Pointer to the argument value.

Description

Pushes a 4–byte Unicode string onto the argument stack.

Return Values for CachePushExStrH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.91 CachePushFunc

Variants: **CachePushFuncW**, **CachePushFuncH**

```
int CachePushFunc(unsigned int * rflag, int tlen, const Callin_char_t * tptr,
                  int nlen, const Callin_char_t * nptr)
```

Arguments

rflag	Routine flags for use by **CacheExtFun**.
tlen	Tag name length (characters), where 0 means that the tag name is null ("").
tptr	Pointer to a tag name. If *tlen* == 0, then *tagptr* is unused and `(void *) 0` may be used as the pointer value.
nlen	Routine name length (characters), where 0 means that the routine name is null ("") and the current routine name is used.
nptr	Pointer to routine name. If *nlen* == 0, then *nptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes an extrinsic function reference onto the argument stack.

Return Values for CachePushFunc

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.92 CachePushFuncH

Variants: **CachePushFunc**, **CachePushFuncW**

```
int CachePushFuncH(unsigned int * rflag, int tlen, const wchar_t * tptr,
                   int nlen, const wchar_t * nptr)
```

Arguments

rflag	Routine flags for use by **CacheExtFun**.
tlen	Tag name length (characters), where 0 means that the tag name is null ("").
tptr	Pointer to a tag name. If *tlen* == 0, then *tptr* is unused and `(void *) 0` may be used as the pointer value.
nlen	Routine name length (characters), where 0 means that the routine name is null ("") and the current routine name is used.
nptr	Pointer to routine name. If *nlen* == 0, then *nptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes a 4-byte Unicode extrinsic function reference onto the argument stack.

Return Values for CachePushFuncH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.93 CachePushFuncW

Variants: **CachePushFunc**, **CachePushFuncH**

```
int CachePushFuncW(unsigned int * rflag, int tlen, const unsigned short * tptr,
                   int nlen, const unsigned short * nptr)
```

Arguments

rflag	Routine flags for use by **CacheExtFun**.
tlen	Tag name length (characters), where 0 means that the tag name is null ("").
tptr	Pointer to a tag name. If *tlen* == 0, then *tptr* is unused and `(void *) 0` may be used as the pointer value.
nlen	Routine name length (characters), where 0 means that the routine name is null ("") and the current routine name is used.
nptr	Pointer to routine name. If *nlen* == 0, then *nptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes a 2-byte Unicode extrinsic function reference onto the argument stack.

Return Values for CachePushFuncW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.94 CachePushFuncX

Variants: **CachePushFuncXW**, **CachePushFuncXH**

```
int CachePushFuncX(unsigned int * rflag, int tlen, const Callin_char_t * tptr, int off,
                   int elen, const Callin_char_t * eptr,
                   int nlen, const Callin_char_t * nptr)
```

Arguments

rflag	Routine flags for use by **CacheExtFun**.
tlen	Tag name length (characters), where 0 means that the tag name is null ("").
tptr	Pointer to a tag name. If *tlen* == 0, then *tptr* is unused and `(void *) 0` may be used as the pointer value.
off	Line offset from specified tag, where 0 means that there is no offset.
elen	Environment name length (characters), where 0 means that there is no environment specified and that the function uses the current environment.
eptr	Pointer to environment name. If *elen* == 0, then *eptr* is unused and `(void *) 0` may be used as the pointer value.
nlen	Routine name length (characters), where 0 means that the routine name is null ("") and the current routine name is used.
nptr	Pointer to routine name. If *nlen* == 0, then *nptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes an extended extrinsic function reference onto the argument stack.

Return Values for CachePushFuncX

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.95 CachePushFuncXH

Variants: **CachePushFuncX**, **CachePushFuncXW**

```
int CachePushFuncXH(unsigned int * rflag, int tlen, const wchar_t * tptr, int off,
                    int elen, const wchar_t * eptr, int nlen, const wchar_t * nptr)
```

Arguments

rflag	Routine flags for use by **CacheExtFun**.
tlen	Tag name length (characters), where 0 means that the tag name is null ("").
tptr	Pointer to a tag name. If *tlen* == 0, then *tptr* is unused and `(void *) 0` may be used as the pointer value.
off	Line offset from specified tag, where 0 means that there is no offset.
elen	Environment name length (characters), where 0 means that there is no environment specified and that the function uses the current environment.
eptr	Pointer to environment name. If *elen* == 0, then *eptr* is unused and `(void *) 0` may be used as the pointer value.
nlen	Routine name length (characters), where 0 means that the routine name is null ("") and the current routine name is used.
nptr	Pointer to routine name. If *nlen* == 0, then *nptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes a 4-byte Unicode extended function routine reference onto the argument stack.

Return Values for CachePushFuncXH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.96 CachePushFuncXW

Variants: **CachePushFuncX**, **CachePushFuncXH**

```
int CachePushFuncXW(unsigned int * rflag, int tlen, const unsigned short * tptr, int off,
                    int elen, const unsigned short * eptr,
                    int nlen, const unsigned short * nptr)
```

Arguments

rflag	Routine flags for use by **CacheExtFun**.
tlen	Tag name length (characters), where 0 means that the tag name is null ("").
tptr	Pointer to a tag name. If *tlen* == 0, then *tptr* is unused and `(void *) 0` may be used as the pointer value.
off	Line offset from specified tag, where 0 means that there is no offset.
elen	Environment name length (characters), where 0 means that there is no environment specified and that the function uses the current environment.
eptr	Pointer to environment name. If *elen* == 0, then *eptr* is unused and `(void *) 0` may be used as the pointer value.
nlen	Routine name length (characters), where 0 means that the routine name is null ("") and the current routine name is used.
nptr	Pointer to routine name. If *nlen* == 0, then *nptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes a 2-byte Unicode extended function routine reference onto the argument stack.

Return Values for CachePushFuncXW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.97 CachePushGlobal

Variants: **CachePushGlobalW**, **CachePushGlobalH**

```
int CachePushGlobal(int nlen, const Callin_char_t * nptr)
```

Arguments

nlen	Global name length (characters).
nptr	Pointer to global name.

Description

Pushes a global reference onto the argument stack.

Return Values for CachePushGlobal

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.98 CachePushGlobalH

Variants: **CachePushGlobal**, **CachePushGlobalW**

```
intCachePushGlobalH(int nlen, const wchar_t * nptr)
```

Arguments

nlen	Global name length (characters).
nptr	Pointer to global name.

Description

Pushes a 4-byte Unicode global reference onto the argument stack.

Return Values for CachePushGlobalH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.99 CachePushGlobalW

Variants: **CachePushGlobal**, **CachePushGlobalH**

```
int CachePushGlobalW(int nlen, const unsigned short * nptr)
```

Arguments

nlen	Global name length (characters).
nptr	Pointer to global name.

Description

Pushes a 2-byte Unicode global reference onto the argument stack.

Return Values for CachePushGlobalW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.100 CachePushGlobalX

Variants: **CachePushGlobalXW**, **CachePushGlobalXH**

```
int CachePushGlobalX(int nlen, const Callin_char_t * nptr,
                     int elen, const Callin_char_t * eptr)
```

Arguments

nlen	Global name length (characters).
nptr	Pointer to global name.
elen	Environment name length (characters), where 0 means that there is no environment specified and that the function uses the current environment.
eptr	Pointer to environment name. If *elen* == 0, then *eptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes an extended global reference onto the argument stack.

Return Values for CachePushGlobalX

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.101 CachePushGlobalXH

Variants: **CachePushGlobalX**, **CachePushGlobalXW**

```
int CachePushGlobalXH(int nlen, const wchar_t * nptr, int elen, const wchar_t * eptr)
```

Arguments

nlen	Global name length (characters).
nptr	Pointer to global name.
elen	Environment name length (characters), where 0 means that there is no environment specified and that the function uses the current environment.
eptr	Pointer to environment name. If *elen* == 0, then *eptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes a 4-byte Unicode extended global reference onto the argument stack.

Return Values for CachePushGlobalXH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTAC	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.102 CachePushGlobalXW

Variants: **CachePushGlobalX**, **CachePushGlobalXH**

```
int CachePushGlobalXW(int nlen, const unsigned short * nptr,
                      int elen, const unsigned short * eptr)
```

Arguments

nlen	Global name length (characters).
nptr	Pointer to global name.
elen	Environment name length (characters), where 0 means that there is no environment specified and that the function uses the current environment.
eptr	Pointer to environment name. If *elen* == 0, then *eptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes a 2-byte Unicode extended global reference onto the argument stack.

Return Values for CachePushGlobalXW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTAC	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.103 CachePushIEEEDbl

```
int CachePushIEEEDbl(double num)
```

Arguments

num	Double value.

Description

Pushes an IEEE double onto the argument stack.

Return Values for CachePushIEEEDbl

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_SUCCESS	The operation was successful.

4.104 CachePushInt

```
int CachePushInt(int num)
```

Arguments

num	Integer value.

Description

Pushes an integer onto the argument stack.

Return Values for CachePushInt

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_SUCCESS	The operation was successful.

4.105 CachePushInt64

```
int CachePushInt64(long long num)
```

Arguments

num	long long value.

Description

Pushes a 64–bit (long long) value onto the argument stack.

Return Values for CachePushInt64

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_SUCCESS	The operation was successful.

4.106 CachePushList

```
int CachePushList(int len, const Callin_char_t * ptr)
```

Arguments

len	Number of characters in string.
ptr	Pointer to string.

Description

Converts a $LIST object and pushes it onto the argument stack. String elements are copied or translated as appropriate depending on whether this is a Unicode or 8-bit version.

Return Values for CachePushList

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a string element.

4.107 CachePushLock

Variants: **CachePushLockW**, **CachePushLockH**

```
int CachePushLock(int nlen, const Callin_char_t * nptr)
```

Arguments

nlen	Length (in bytes) of lock name.
nptr	Pointer to lock name.

Description

Initializes a **CacheAcquireLock** command by pushing the lock name on the argument stack.

Return Values for CachePushLock

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.108 CachePushLockH

Variants: **CachePushLock**, **CachePushLockW**

```
int CachePushLockH(int nlen, const wchar_t * nptr)
```

Arguments

nlen	Length (number of 2–byte or 4–byte characters) of lock name.
nptr	Pointer to lock name.

Description

Initializes a **CacheAcquireLock** command by pushing the lock name on the argument stack.

Return Values for CachePushLockH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.109 CachePushLockW

Variants: **CachePushLock**, **CachePushLockH**

```
int CachePushLockW(int nlen, const unsigned short * nptr)
```

Arguments

nlen	Length (number of 2–byte characters) of lock name.
nptr	Pointer to lock name.

Description

Initializes a **CacheAcquireLock** command by pushing the lock name on the argument stack.

Return Values for CachePushLockW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.110 CachePushLockX

Variants: **CachePushLockXW**, **CachePushLockXH**

```
int CachePushLockX(int nlen, const Callin_char_t * nptr, int elen, const Callin_char_t * eptr)
```

Arguments

nlen	Length (number of 8–bit characters) of lock name.
nptr	Pointer to lock name.
elen	Environment name length (characters), where 0 means that there is no environment specified and that the function uses the current environment. Name must be of the form `<Namespace>^[<system>]^<directory>`
eptr	Pointer to environment name. If *elen* == 0, then *eptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Initializes a **CacheAcquireLock** command by pushing the lock name and an environment string on the argument stack.

Return Values for CachePushLockX

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.111 CachePushLockXH

Variants: **CachePushLockX**, **CachePushLockXW**

```
int CachePushLockXH(int nlen, const wchar_t * nptr, int elen, const wchar_t * eptr)
```

Arguments

nlen	Length (number of 2–byte or 4–byte characters) of lock name.
nptr	Pointer to lock name.
elen	Environment name length (characters), where 0 means that there is no environment specified and that the function uses the current environment. Name must be of the form `<Namespace>^[<system>]^<directory>`
eptr	Pointer to environment name. If *elen* == 0, then *eptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Initializes a **CacheAcquireLock** command by pushing the lock name and an environment string on the argument stack.

Return Values for CachePushLockXH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.112 CachePushLockXW

Variants: **CachePushLockX**, **CachePushLockXH**

```
int CachePushLockXW(int nlen, const unsigned short * nptr, int elen, const unsigned short * eptr)
```

Arguments

nlen	Length (number of 2–byte characters) of lock name.
nptr	Pointer to lock name.
elen	Environment name length (characters), where 0 means that there is no environment specified and that the function uses the current environment. Name must be of the form `<Namespace>^[<system>]^<directory>`
eptr	Pointer to environment name. If *elen* == 0, then *eptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Initializes a **CacheAcquireLock** command by pushing the lock name and an environment string on the argument stack.

Return Values for CachePushLockXW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.113 CachePushMethod

Variants: **CachePushMethodW**, **CachePushMethodH**

```
int CachePushMethod(unsigned int oref, int mlen, const Callin_char_t * mptr, int flg)
```

Arguments

oref	Object reference.
mlen	Method name length (characters).
mptr	Pointer to method name.
flg	Specifies whether the method will return a value. If the method returns a value, this flag must be set to `1` in order to retrieve it. The method must return a value via **Quit** with an argument. Set this parameter to `0` if no value will be returned.

Description

Pushes an instance method reference onto the argument stack.

Return Values for CachePushMethod

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_BADARG	Invalid call argument.
CACHE_SUCCESS	The operation was successful.

4.114 CachePushMethodH

Variants: **CachePushMethod**, **CachePushMethodW**

```
int CachePushMethodH(unsigned int oref, int mlen, const wchar_t * mptr, int flg)
```

Arguments

oref	Object reference.
mlen	Method name length (characters).
mptr	Pointer to method name.
flg	Specifies whether the method will return a value. If the method returns a value, this flag must be set to `1` in order to retrieve it. The method must return a value via **Quit** with an argument. Set this parameter to `0` if no value will be returned.

Description

Pushes a 4-byte Unicode instance method reference onto the argument stack.

Return Values for CachePushMethodH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_BADARG	Invalid call argument.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.115 CachePushMethodW

Variants: **CachePushMethod**, **CachePushMethodH**

```
int CachePushMethodW(unsigned int oref, int mlen, const unsigned short * mptr, int flg)
```

Arguments

oref	Object reference.
mlen	Method name length (characters).
mptr	Pointer to method name.
flg	Specifies whether the method will return a value. If the method returns a value, this flag must be set to `1` in order to retrieve it. The method must return a value via **Quit** with an argument. Set this parameter to `0` if no value will be returned.

Description

Pushes a 2-byte Unicode instance method reference onto the argument stack.

Return Values for CachePushMethodW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_BADARG	Invalid call argument.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.116 CachePushOref

```
int CachePushOref(unsigned int oref)
```

Arguments

oref	Object reference.

Description

Pushes an OREF onto the argument stack.

Return Values for CachePushOref

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERBADOREF	Invalid OREF.
CACHE_SUCCESS	The operation was successful.

4.117 CachePushProperty

Variants: **CachePushPropertyW**, **CachePushPropertyH**

```
int CachePushProperty(unsigned int oref, int plen, const Callin_char_t * pptr)
```

Arguments

oref	Object reference.
plen	Property name length (characters).
pptr	Pointer to property name.

Description

Pushes a property reference onto the argument stack.

Return Values for CachePushProperty

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_BADARG	Invalid call argument.
CACHE_SUCCESS	The operation was successful.

4.118 CachePushPropertyH

Variants: **CachePushProperty**, **CachePushPropertyW**

```
int CachePushPropertyH(unsigned int oref, int plen, const wchar_t * pptr)
```

Arguments

oref	Object reference.
plen	Property name length (characters).
pptr	Pointer to property name.

Description

Pushes a 4-byte Unicode property reference onto the argument stack.

Return Values for CachePushPropertyH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_BADARG	Invalid call argument.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.119 CachePushPropertyW

Variants: **CachePushProperty**, **CachePushPropertyH**

```
int CachePushPropertyW(unsigned int oref, int plen, const unsigned short * pptr)
```

Arguments

oref	Object reference.
plen	Property name length (characters).
pptr	Pointer to property name.

Description

Pushes a 2-byte Unicode property reference onto the argument stack.

Return Values for CachePushPropertyW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_BADARG	Invalid call argument.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.120 CachePushPtr

```
int CachePushPtr(void * ptr)
```

Arguments

ptr	Generic pointer.

Description

Pushes a pointer onto the argument stack in internal format.

Return Values for CachePushPtr

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.121 CachePushRtn

Variants: **CachePushRtnW**, **CachePushRtnH**

```
int CachePushRtn(unsigned int * rflag, int tlen, const Callin_char_t * tptr,
                 int nlen, const Callin_char_t * nptr)
```

Arguments

rflag	Routine flags for use by **CacheDoRtn**
tlen	Tag name length (characters), where 0 means that the tag name is null ("").
tptr	Pointer to a tag name. If *tlen* == 0, then *tptr* is unused and `(void *) 0` may be used as the pointer value.
nlen	Routine name length (characters), where 0 means that the routine name is null ("") and the current routine name is used.
nptr	Pointer to routine name. If *nlen* == 0, then *nptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes a routine reference onto the argument stack. See CachePushRtnX for a version that takes all arguments. This is a short form that only takes a tag name and a routine name.

Return Values for CachePushRtn

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.122 CachePushRtnH

Variants: **CachePushRtn**, **CachePushRtnW**

```
int CachePushRtnH(unsigned int * rflag, int tlen, const wchar_t * tptr,
                  int nlen, const wchar_t * nptr)
```

Arguments

rflag	Routine flags for use by **CacheDoRtn**
tlen	Tag name length (characters), where 0 means that the tag name is null ("").
tptr	Pointer to a tag name. If *tlen* == 0, then *tptr* is unused and `(void *) 0` may be used as the pointer value.
nlen	Routine name length (characters), where 0 means that the routine name is null ("") and the current routine name is used.
nptr	Pointer to routine name. If *nlen* == 0, then *nptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes a 4–byte Unicode routine reference onto the argument stack. See CachePushRtnXH for a version that takes all arguments. This is a short form that only takes a tag name and a routine name.

Return Values for CachePushRtnH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.123 CachePushRtnW

Variants: **CachePushRtn**, **CachePushRtnH**

```
int CachePushRtnW(unsigned int * rflag, int tlen, const unsigned short * tptr,
                  int nlen, const unsigned short * nptr)
```

Arguments

rflag	Routine flags for use by **CacheDoRtn**
tlen	Tag name length (characters), where 0 means that the tag name is null ("").
tptr	Pointer to a tag name. If *tlen* == 0, then *tptr* is unused and `(void *) 0` may be used as the pointer value.
nlen	Routine name length (characters), where 0 means that the routine name is null ("") and the current routine name is used.
nptr	Pointer to routine name. If *nlen* == 0, then *nptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes a 2–byte Unicode routine reference onto the argument stack. See CachePushRtnXW for a version that takes all arguments. This is a short form that only takes a tag name and a routine name.

Return Values for CachePushRtnW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.124 CachePushRtnX

Variants: **CachePushRtnXW**, **CachePushRtnXH**

```
int CachePushRtnX(unsigned int * rflag, int tlen, const Callin_char_t * tptr,
                  int off, int elen, const Callin_char_t * eptr,
                  int nlen, const Callin_char_t * nptr)
```

Arguments

rflag	Routine flags for use by **CacheDoRtn**
tlen	Tag name length (characters), where 0 means that the tag name is null ("").
tptr	Pointer to a tag name. If *tlen* == 0, then *tptr* is unused and `(void *) 0` may be used as the pointer value.
off	Line offset from specified tag, where 0 means that there is no offset.
elen	Environment name length (characters), where 0 means that there is no environment specified and that the function uses the current environment.
eptr	Pointer to environment name. If *elen* == 0, then *eptr* is unused and `(void *) 0` may be used as the pointer value.
nlen	Routine name length (characters), where 0 means that the routine name is null ("") and the current routine name is used.
nptr	Pointer to routine name. If *nlen* == 0, then *nptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes an extended routine reference onto the argument stack. See CachePushRtn for a short form that only takes a tag name and a routine name.

Return Values for CachePushRtnX

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.125 CachePushRtnXH

Variants: **CachePushRtnX**, **CachePushRtnXW**

```
int CachePushRtnXH(unsigned int * rflag, int tlen, const wchar_t * tptr,
                   int off, int elen, const wchar_t * eptr,
                   int nlen, const wchar_t * nptr)
```

Arguments

rflag	Routine flags for use by **CacheDoRtn**
tlen	Tag name length (characters), where 0 means that the tag name is null ("").
tptr	Pointer to a tag name. If *tlen* == 0, then *tptr* is unused and `(void *) 0` may be used as the pointer value.
off	Line offset from specified tag, where 0 means that there is no offset.
elen	Environment name length (characters), where 0 means that there is no environment specified and that the function uses the current environment.
eptr	Pointer to environment name. If *elen* == 0, then *eptr* is unused and `(void *) 0` may be used as the pointer value.
nlen	Routine name length (characters), where 0 means that the routine name is null ("") and the current routine name is used.
nptr	Pointer to routine name. If *nlen* == 0, then *nptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes a 4–byte Unicode extended routine reference onto the argument stack. See CachePushRtnH for a short form that only takes a tag name and a routine name.

Return Values for CachePushRtnXH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.126 CachePushRtnXW

Variants: **CachePushRtnX**, **CachePushRtnXH**

```
int CachePushRtnXW(unsigned int * rflag, int tlen, const unsigned short * tptr,
                   int off, int elen, const unsigned short * eptr,
                   int nlen, const unsigned short * nptr)
```

Arguments

rflag	Routine flags for use by **CacheDoRtn**
tlen	Tag name length (characters), where 0 means that the tag name is null ("").
tptr	Pointer to a tag name. If *tlen* == 0, then *tptr* is unused and `(void *) 0` may be used as the pointer value.
off	Line offset from specified tag, where 0 means that there is no offset.
elen	Environment name length (characters), where 0 means that there is no environment specified and that the function uses the current environment.
eptr	Pointer to environment name. If *elen* == 0, then *eptr* is unused and `(void *) 0` may be used as the pointer value.
nlen	Routine name length (characters), where 0 means that the routine name is null ("") and the current routine name is used.
nptr	Pointer to routine name. If *nlen* == 0, then *nptr* is unused and `(void *) 0` may be used as the pointer value.

Description

Pushes a 2–byte Unicode extended routine reference onto the argument stack. See CachePushRtnW for a short form that only takes a tag name and a routine name.

Return Values for CachePushRtnXW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.127 CachePushStr

Variants: **CachePushStrW**, **CachePushStrH**

```
int CachePushStr(int len, const Callin_char_t * ptr)
```

Arguments

len	Number of characters in string.
ptr	Pointer to string.

Description

Pushes a byte string onto the argument stack.

Return Values for CachePushStr

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.128 CachePushStrH

Variants: **CachePushStr**, **CachePushStrW**

```
int CachePushStrH(int len, const wchar_t * ptr)
```

Arguments

len	Number of characters in string.
ptr	Pointer to string.

Description

Pushes a 4-byte Unicode string onto the argument stack.

Return Values for CachePushStrH

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.129 CachePushStrW

Variants: **CachePushStr**, **CachePushStrH**

```
int CachePushStrW(int len, const unsigned short * ptr)
```

Arguments

len	Number of characters in string.
ptr	Pointer to string.

Description

Pushes a 2-byte Unicode string onto the argument stack.

Return Values for CachePushStrW

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.

4.130 CachePushUndef

```
int CachePushUndef()
```

Description

Pushes an Undefined value on the argument stack. The value is interpreted as an omitted function argument.

Return Values for CachePushUndef

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_SUCCESS	The operation was successful.

4.131 CacheReleaseAllLocks

```
int CacheReleaseAllLocks( )
```

Description

Performs an argumentless Cache LOCK command to remove all locks currently held by the process.

Return Values for CacheReleaseAllLocks

CACHE_SUCCESS	The operation was successful.

4.132 CacheReleaseLock

```
int CacheReleaseLock(int nsub, int flg)
```

Arguments

nsub	Number of subscripts in the lock reference.
flg	Modifiers to the lock command. Valid values are one or both of CACHE_IMMEDIATE_RELEASE and CACHE_SHARED_LOCK.

Description

Executes a Cache LOCK command to decrement the lock count for the specified lock name. This command will only release one incremental lock at a time.

Return Values for CacheReleaseLock

CACHE_FAILURE	An unexpected error has occurred.
CACHE_SUCCESS	Successful lock.

4.133 CacheSecureStartA

Variants: **CacheSecureStartW**, **CacheSecureStartH**

```
int CacheSecureStartA(CACHE_ASTRP username, CACHE_ASTRP password, CACHE_ASTRP exename,
                      unsigned long flags, int tout, CACHE_ASTRP prinp, CACHE_ASTRP prout)
```

Arguments

username	Username to authenticate. Use NULL to authenticate as UnknownUseror OS authentication or kerberos credentials cache.
password	Password to authenticate with. Use NULL to authenticate as UnknownUser or OS authentication or kerberos credentials cache.
exename	Callin executable name (or other process identifier). This user-defined string will show up in JOBEXAM and in audit records. NULL is a valid value.
flags	One or more of the terminal settings listed below.
tout	The timeout specified in seconds. Default is 0. If 0 is specified, the timeout will never expire. The timeout applies only to waiting for an available partition, not the time associated with initializing the partition, waiting for internal resources, opening the principal input and output devices, etc.
prinp	String that defines the principal input device for Caché. An empty string (*prinp.len* == 0) implies using the standard input device for the process. A NULL pointer (`(void *) 0`) implies using the NULL device.
prout	String that defines the principal output device for Caché. An empty string (*prout.len* == 0) implies using the standard output device for the process. A NULL pointer (`(void *) 0`) implies using the NULL device.

Description

Calls into Cache to set up a Cache process..

The input and output devices (*prinp* and *prout*) are opened when this command is executed, not deferred until the first I/O operation. By contrast, normally when you initiate a Caché connection with the cache command, Caché does not open the principal input or output device until it is first used.

Valid values for the *flags* variable are:

- CACHE_PROGMODE — Caché should treat the connection as one in Programmer mode, rather than the Application mode. This means that distinct errors are reported to the calling function and the connection remains active. (By default, a Callin connection is like execution of a routine in application mode. Any runtime error detected by Caché results in closing the connection and returning error CACHE_CONBROKEN for both the current operation and any subsequent attempts to use Callin without establishing a new connection.)
- CACHE_TTALL — Default. Caché should initialize the terminal's settings and restore them across each call into, and return from, the interface.
- CACHE_TTCALLIN — Caché should initialize the terminal each time it is called but should restore it only when **CacheEnd** is called or the connection is broken.
- CACHE_TTSTART — Caché should initialize the terminal when the connection is formed and reset it when the connection is terminated.
- CACHE_TTNEVER — Caché should not alter the terminal's settings.
- CACHE_TTNONE — Caché should not do any output or input from the principal input/output devices. This is equivalent to specifying the null device for principal input and principal output. **Read** commands from principal input generate an <ENDOFFILE> error and **Write** command to principal output are ignored.
- CACHE_TTNOUSE — This flag is allowed with CACHE_TTALL, CACHE_TTCALLIN, and CACHE_TTSTART. It is implicitly set by the flags CACHE_TTNEVER and CACHE_TTNONE. It indicates that Caché **Open** and **Use** commands are not allowed to alter the terminal, subsequent to the initial open of principal input and principal output.

Return Values for CacheSecureStartA

CACHE_ACCESSDENIED	Authentication has failed. Check the audit log for the real authentication error.
CACHE_ALREADYCON	Connection already existed. Returned if you call **CacheSecureStartH** from a **$ZF** function.
CACHE_CHANGEPASSWORD	Password change required. This return value is only returned if you are using Caché authentication.
CACHE_CONBROKEN	Connection was formed and then broken, and **CacheEnd** has not been called to clean up.
CACHE_FAILURE	An unexpected error has occurred.
CACHE_STRTOOLONG	*prinp* or *prout* is too long.
CACHE_SUCCESS	Connection formed.

The flags parameter(s) convey information about how your C program will behave and how you want Caché to set terminal characteristics. The safest, but slowest, route is to have Caché set and restore terminal settings for each call into ObjectScript. However, you can save ObjectScript overhead by handling more of that yourself, and collecting only information that matters to your program. The parameter CACHE_TTNEVER requires the least overhead.

4.134 CacheSecureStartH

Variants: **CacheSecureStartA**, **CacheSecureStartW**

```
int CacheSecureStartH(CACHEHSTRP username, CACHEHSTRP password, CACHEHSTRP exename,
                      unsigned long flags, int tout, CACHEHSTRP prinp, CACHEHSTRP prout)
```

Arguments

username	Username to authenticate. Use NULL to authenticate as UnknownUseror OS authentication or kerberos credentials cache.
password	Password to authenticate with. Use NULL to authenticate as UnknownUser or OS authentication or kerberos credentials cache.
exename	Callin executable name (or other process identifier). This user-defined string will show up in JOBEXAM and in audit records. NULL is a valid value.
flags	One or more of the terminal settings listed below.
tout	The timeout specified in seconds. Default is 0. If 0 is specified, the timeout will never expire. The timeout applies only to waiting for an available partition, not the time associated with initializing the partition, waiting for internal resources, opening the principal input and output devices, etc.
prinp	String that defines the principal input device for Caché. An empty string (*prinp.len* == 0) implies using the standard input device for the process. A NULL pointer (`(void *) 0`) implies using the NULL device.
prout	String that defines the principal output device for Caché. An empty string (*prout.len* == 0) implies using the standard output device for the process. A NULL pointer (`(void *) 0`) implies using the NULL device.

Description

Calls into Cache to set up a Cache process..

The input and output devices (*prinp* and *prout*) are opened when this command is executed, not deferred until the first I/O operation. By contrast, normally when you initiate a Caché connection with the cache command, Caché does not open the principal input or output device until it is first used.

Valid values for the *flags* variable are:

- CACHE_PROGMODE — Caché should treat the connection as one in Programmer mode, rather than the Application mode. This means that distinct errors are reported to the calling function and the connection remains active. (By default, a Callin connection is like execution of a routine in application mode. Any runtime error detected by Caché results in closing the connection and returning error CACHE_CONBROKEN for both the current operation and any subsequent attempts to use Callin without establishing a new connection.)
- CACHE_TTALL — Default. Caché should initialize the terminal's settings and restore them across each call into, and return from, the interface.
- CACHE_TTCALLIN — Caché should initialize the terminal each time it is called but should restore it only when **CacheEnd** is called or the connection is broken.
- CACHE_TTSTART — Caché should initialize the terminal when the connection is formed and reset it when the connection is terminated.
- CACHE_TTNEVER — Caché should not alter the terminal's settings.

- CACHE_TTNONE — Caché should not do any output or input from the principal input/output devices. This is equivalent to specifying the null device for principal input and principal output. **Read** commands from principal input generate an <ENDOFFILE> error and **Write** command to principal output are ignored.
- CACHE_TTNOUSE — This flag is allowed with CACHE_TTALL, CACHE_TTCALLIN, and CACHE_TTSTART. It is implicitly set by the flags CACHE_TTNEVER and CACHE_TTNONE. It indicates that Caché **Open** and **Use** commands are not allowed to alter the terminal, subsequent to the initial open of principal input and principal output.

Return Values for CacheSecureStartH

CACHE_ACCESSDENIED	Authentication has failed. Check the audit log for the real authentication error.
CACHE_ALREADYCON	Connection already existed. Returned if you call **CacheSecureStartH** from a **$ZF** function.
CACHE_CHANGEPASSWORD	Password change required. This return value is only returned if you are using Caché authentication.
CACHE_CONBROKEN	Connection was formed and then broken, and **CacheEnd** has not been called to clean up.
CACHE_FAILURE	An unexpected error has occurred.
CACHE_STRTOOLONG	*prinp* or *prout* is too long.
CACHE_SUCCESS	Connection formed.

The flags parameter(s) convey information about how your C program will behave and how you want Caché to set terminal characteristics. The safest, but slowest, route is to have Caché set and restore terminal settings for each call into ObjectScript. However, you can save ObjectScript overhead by handling more of that yourself, and collecting only information that matters to your program. The parameter CACHE_TTNEVER requires the least overhead.

4.135 CacheSecureStartW

Variants: **CacheSecureStartA**, **CacheSecureStartH**

```
int CacheSecureStartW(CACHEWSTRP username, CACHEWSTRP password, CACHEWSTRP exename,
                      unsigned long flags, int tout, CACHEWSTRP prinp, CACHEWSTRP prout)
```

Arguments

username	Username to authenticate. Use NULL to authenticate as UnknownUseror OS authentication or kerberos credentials cache.
password	Password to authenticate with. Use NULL to authenticate as UnknownUser or OS authentication or kerberos credentials cache.
exename	Callin executable name (or other process identifier). This user-defined string will show up in JOBEXAM and in audit records. NULL is a valid value.
flags	One or more of the terminal settings listed below.
tout	The timeout specified in seconds. Default is 0. If 0 is specified, the timeout will never expire. The timeout applies only to waiting for an available partition, not the time associated with initializing the partition, waiting for internal resources, opening the principal input and output devices, etc.
prinp	String that defines the principal input device for Caché. An empty string (*prinp.len* == 0) implies using the standard input device for the process. A NULL pointer (`((void *) 0)`) implies using the NULL device.
prout	String that defines the principal output device for Caché. An empty string (*prout.len* == 0) implies using the standard output device for the process. A NULL pointer (`((void *) 0)`) implies using the NULL device.

Description

Calls into Cache to set up a Cache process..

The input and output devices (*prinp* and *prout*) are opened when this command is executed, not deferred until the first I/O operation. By contrast, normally when you initiate a Caché connection with the cache command, Caché does not open the principal input or output device until it is first used.

Valid values for the *flags* variable are:

- CACHE_PROGMODE — Caché should treat the connection as one in Programmer mode, rather than the Application mode. This means that distinct errors are reported to the calling function and the connection remains active. (By default, a Callin connection is like execution of a routine in application mode. Any runtime error detected by Caché results in closing the connection and returning error CACHE_CONBROKEN for both the current operation and any subsequent attempts to use Callin without establishing a new connection.)
- CACHE_TTALL — Default. Caché should initialize the terminal's settings and restore them across each call into, and return from, the interface.
- CACHE_TTCALLIN — Caché should initialize the terminal each time it is called but should restore it only when **CacheEnd** is called or the connection is broken.
- CACHE_TTSTART — Caché should initialize the terminal when the connection is formed and reset it when the connection is terminated.
- CACHE_TTNEVER — Caché should not alter the terminal's settings.
- CACHE_TTNONE — Caché should not do any output or input from the principal input/output devices. This is equivalent to specifying the null device for principal input and principal output. **Read** commands from principal input generate an <ENDOFFILE> error and **Write** command to principal output are ignored.
- CACHE_TTNOUSE — This flag is allowed with CACHE_TTALL, CACHE_TTCALLIN, and CACHE_TTSTART. It is implicitly set by the flags CACHE_TTNEVER and CACHE_TTNONE. It indicates that Caché **Open** and **Use** commands are not allowed to alter the terminal, subsequent to the initial open of principal input and principal output.

Return Values for CacheSecureStartW

CACHE_ACCESSDENIED	Authentication has failed. Check the audit log for the real authentication error.
CACHE_ALREADYCON	Connection already existed. Returned if you call **CacheSecureStartH** from a **$ZF** function.
CACHE_CHANGEPASSWORD	Password change required. This return value is only returned if you are using Caché authentication.
CACHE_CONBROKEN	Connection was formed and then broken, and **CacheEnd** has not been called to clean up.
CACHE_FAILURE	An unexpected error has occurred.
CACHE_STRTOOLONG	*prinp* or *prout* is too long.
CACHE_SUCCESS	Connection formed.

The flags parameter(s) convey information about how your C program will behave and how you want Caché to set terminal characteristics. The safest, but slowest, route is to have Caché set and restore terminal settings for each call into ObjectScript. However, you can save ObjectScript overhead by handling more of that yourself, and collecting only information that matters to your program. The parameter CACHE_TTNEVER requires the least overhead.

4.136 CacheSetDir

```
int CacheSetDir(char * dir)
```

Arguments

dir	Pointer to the directory name string.

Description

Dynamically sets the name of the manager's directory (CacheSys\Mgr) at runtime. On Windows, the shared library version of Caché requires the use of this function to identify the managers directory for the installation.

Return Values for CacheSetDir

CACHE_FAILURE	Returns if called from a **$ZF** function (rather than from within a Callin executable).
CACHE_SUCCESS	Control function performed.

4.137 CacheSetProperty

```
int CacheSetProperty( )
```

Description

Stores the value of the property defined by **CachePushProperty**. The value must be pushed onto the argument stack before this call.

Return Values for CacheSetProperty

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_SUCCESS	The operation was successful.

4.138 CacheSignal

```
int CacheSignal(int signal)
```

Arguments

signal	The operating system's signal value.

Description

Passes on signals caught by user's program to Caché.

This function is very similar to **CacheAbort**, but allows passing of any known signal value from a thread or user side of the connection to the Caché side, for whatever action might be appropriate. For example, this could be used to pass signals intercepted in a user-defined signal handler on to Caché.

Example

```
rc = CacheSignal(CTRL_C_EVENT); // Windows response to Ctrl-C
rc = CacheSignal(CTRL_C_EVENT); // UNIX response to Ctrl-C
```

Return Values for CacheSignal

CACHE_CONBROKEN	Connection has been broken.
CACHE_NOCON	No connection has been established.
CACHE_NOTINCACHE	The Callin partner is not in Caché at this time.
CACHE_SUCCESS	Connection formed.

4.139 CacheSPCReceive

```
int CacheSPCReceive(int * lenp, Callin_char_t * ptr)
```

Arguments

lenp	Maximum length to receive. Modified on return to indicate number of bytes actually received.
ptr	Pointer to buffer that will receive message. Must be at least *lenp* bytes.

Description

Receive single-process-communication message. The current device must be a TCP device opened in SPC mode, or CACHE_ERFUNCTION will be returned.

Return Values for CacheSPCReceive

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERFUNCTION	Current device is not TCP device or is not connected.
CACHE_SUCCESS	The operation was successful.

4.140 CacheSPCSend

```
int CacheSPCSend(int len, const Callin_char_t * ptr)
```

Arguments

len	Length of message in bytes.
ptr	Pointer to string containing message.

Description

Send a single-process-communication message. The current device must be a TCP device opened in SPC mode, or CACHE_ERFUNCTION will be returned.

Return Values for CacheSPCSend

CACHE_CONBROKEN	Connection has been closed due to a serious error.
CACHE_NOCON	No connection has been established.
CACHE_ERSYSTEM	Either the Caché engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
CACHE_ERFUNCTION	Current device is not TCP device or is not connected.
CACHE_ERARGSTACK	Argument stack overflow.
CACHE_ERSTRINGSTACK	String stack overflow.
CACHE_SUCCESS	The operation was successful.
Any Caché error	From translating a name.

4.141 CacheStartA

Variants: **CacheStartW**, **CacheStartH**

```
int CacheStartA(unsigned long flags, int tout, CACHE_ASTRP prinp, CACHE_ASTRP prout)
```

Arguments

flags	One or more of the values listed in the description below.
tout	The timeout specified in seconds. Default is 0. If 0 is specified, the timeout will never expire. The timeout applies only to waiting for an available partition, not the time associated with initializing the partition, waiting for internal resources, opening the principal input and output devices, etc.
prinp	String that defines the principal input device for Caché. An empty string (*prinp.len* == 0) implies using the standard input device for the process. A NULL pointer (`(void *) 0`) implies using the NULL device.
prout	String that defines the principal output device for Caché. An empty string (*prout.len* == 0) implies using the standard output device for the process. A NULL pointer (`(void *) 0`) implies using the NULL device.

Description

Calls into Caché to set up a Caché process.

The input and output devices (*prinp* and *prout*) are opened when this command is executed, not deferred until the first I/O operation. By contrast, normally when you initiate a Caché connection with the cache command, Caché does not open the principal input or output device until it is first used.

Valid values for the *flags* variable are:

- CACHE_PROGMODE — Caché should treat the connection as one in Programmer mode, rather than the Application mode. This means that distinct errors are reported to the calling function and the connection remains active. (By default, a Callin connection is like execution of a routine in application mode. Any runtime error detected by Caché results in closing the connection and returning error CACHE_CONBROKEN for both the current operation and any subsequent attempts to use Callin without establishing a new connection.)
- CACHE_TTALL — Default. Caché should initialize the terminal's settings and restore them across each call into, and return from, the interface.
- CACHE_TTCALLIN — Caché should initialize the terminal each time it is called but should restore it only when **CacheEnd** is called or the connection is broken.
- CACHE_TTSTART — Caché should initialize the terminal when the connection is formed and reset it when the connection is terminated.
- CACHE_TTNEVER — Caché should not alter the terminal's settings.
- CACHE_TTNONE — Caché should not do any output or input from the principal input/output devices. This is equivalent to specifying the null device for principal input and principal output. **Read** commands from principal input generate an <ENDOFFILE> error and **Write** command to principal output are ignored.
- CACHE_TTNOUSE — This flag is allowed with CACHE_TTALL, CACHE_TTCALLIN, and CACHE_TTSTART. It is implicitly set by the flags CACHE_TTNEVER and CACHE_TTNONE. It indicates that Caché **Open** and **Use** commands are not allowed to alter the terminal, subsequent to the initial open of principal input and principal output.

Return Values for CacheStartA

CACHE_ALREADYCON	Connection already existed. Returned if you call **CacheStartA** from a **$ZF** function.
CACHE_CONBROKEN	Connection was formed and then broken, and **CacheEndA** has not been called to clean up.
CACHE_FAILURE	An unexpected error has occurred.
CACHE_STRTOOLONG	*prinp* or *prout* is too long.
CACHE_SUCCESS	Connection formed.

The flags parameter(s) convey information about how your C program will behave and how you want Caché to set terminal characteristics. The safest, but slowest, route is to have Caché set and restore terminal settings for each call into ObjectScript. However, you can save ObjectScript overhead by handling more of that yourself, and collecting only information that matters to your program. The parameter CACHE_TTNEVER requires the least overhead.

Example

A Caché process is started. The terminal is reset after each interface Callin function. The start fails if a partition is not allocated within 20 seconds. The file dobackup is used for input. It contains an ObjectScript script for a Caché backup. Output appears on the terminal.

```
CACHE_ASTR inpdev;
CACHE_ASTR outdev;
int rc;

strcpy(inpdev.str, "[BATCHDIR]dobackup");
inpdev.len = strlen(inpdev.str);
strcpy(outdev.str,"");
outdev.len = strlen(outdev.str);
rc = CacheStartA(CACHE_TTALL|CACHE_TTNOUSE,0,inpdev,outdev);
```

4.142 CacheStartH

Variants: **CacheStartA**, **CacheStartW**

```
int CacheStartH(unsigned long flags,int tout,CACHEHSTRP prinp,CACHEHSTRP prout)
```

Arguments

flags	One or more of the values listed in the description below.
tout	The timeout specified in seconds. Default is 0. If 0 is specified, the timeout will never expire. The timeout applies only to waiting for an available partition, not the time associated with initializing the partition, waiting for internal resources, opening the principal input and output devices, etc.
prinp	String that defines the principal input device for Caché. An empty string (*prinp.len* == 0) implies using the standard input device for the process. A NULL pointer (`(void *) 0`) implies using the NULL device.
prout	String that defines the principal output device for Caché. An empty string (*prout.len* == 0) implies using the standard output device for the process. A NULL pointer (`(void *) 0`) implies using the NULL device.

Description

Calls into Caché to set up a Caché process.

The input and output devices (*prinp* and *prout*) are opened when this command is executed, not deferred until the first I/O operation. By contrast, normally when you initiate a Caché connection with the cache command, Caché does not open the principal input or output device until it is first used.

Valid values for the *flags* variable are:

- CACHE_PROGMODE — Caché should treat the connection as one in Programmer mode, rather than the Application mode. This means that distinct errors are reported to the calling function and the connection remains active. (By default, a Callin connection is like execution of a routine in application mode. Any runtime error detected by Caché results in closing the connection and returning error CACHE_CONBROKEN for both the current operation and any subsequent attempts to use Callin without establishing a new connection.)
- CACHE_TTALL — Default. Caché should initialize the terminal's settings and restore them across each call into, and return from, the interface.
- CACHE_TTCALLIN — Caché should initialize the terminal each time it is called but should restore it only when **CacheEnd** is called or the connection is broken.
- CACHE_TTSTART — Caché should initialize the terminal when the connection is formed and reset it when the connection is terminated.
- CACHE_TTNEVER — Caché should not alter the terminal's settings.
- CACHE_TTNONE — Caché should not do any output or input from the principal input/output devices. This is equivalent to specifying the null device for principal input and principal output. **Read** commands from principal input generate an <ENDOFFILE> error and **Write** command to principal output are ignored.
- CACHE_TTNOUSE — This flag is allowed with CACHE_TTALL, CACHE_TTCALLIN, and CACHE_TTSTART. It is implicitly set by the flags CACHE_TTNEVER and CACHE_TTNONE. It indicates that Caché **Open** and **Use** commands are not allowed to alter the terminal, subsequent to the initial open of principal input and principal output.

Return Values for CacheStartH

CACHE_ALREADYCON	Connection already existed. Returned if you call **CacheStartH** from a **$ZF** function.
CACHE_CONBROKEN	Connection was formed and then broken, and **CacheEndH** has not been called to clean up.
CACHE_FAILURE	An unexpected error has occurred.
CACHE_STRTOOLONG	*prinp* or *prout* is too long.
CACHE_SUCCESS	Connection formed.

The flags parameter(s) convey information about how your C program will behave and how you want Caché to set terminal characteristics. The safest, but slowest, route is to have Caché set and restore terminal settings for each call into ObjectScript. However, you can save ObjectScript overhead by handling more of that yourself, and collecting only information that matters to your program. The parameter CACHE_TTNEVER requires the least overhead.

Example

A Caché process is started. The terminal is reset after each interface Callin function. The start fails if a partition is not allocated within 20 seconds. The file dobackup is used for input. It contains an ObjectScript script for a Caché backup. Output appears on the terminal.

```
inpdev;
outdev;
int rc;

strcpy(inpdev.str, "[BATCHDIR]dobackup");
inpdev.len = strlen(inpdev.str);
strcpy(outdev.str,"");
outdev.len = strlen(outdev.str);
rc = CacheStartH(CACHE_TTALL|CACHE_TTNOUSE,0,inpdev,outdev);
```

4.143 CacheStartW

Variants: **CacheStartA**, **CacheStartH**

```
int CacheStartW(unsigned long flags,int tout,CACHEWSTRP prinp,CACHEWSTRP prout)
```

Arguments

flags	One or more of the values listed in the description below.
tout	The timeout specified in seconds. Default is 0. If 0 is specified, the timeout will never expire. The timeout applies only to waiting for an available partition, not the time associated with initializing the partition, waiting for internal resources, opening the principal input and output devices, etc.
prinp	String that defines the principal input device for Caché. An empty string (*prinp.len* == 0) implies using the standard input device for the process. A NULL pointer (`((void *) 0)`) implies using the NULL device.
prout	String that defines the principal output device for Caché. An empty string (*prout.len* == 0) implies using the standard output device for the process. A NULL pointer (`((void *) 0)`) implies using the NULL device.

Description

Calls into Caché to set up a Caché process.

The input and output devices (*prinp* and *prout*) are opened when this command is executed, not deferred until the first I/O operation. By contrast, normally when you initiate a Caché connection with the cache command, Caché does not open the principal input or output device until it is first used.

Valid values for the *flags* variable are:

- CACHE_PROGMODE — Caché should treat the connection as one in Programmer mode, rather than the Application mode. This means that distinct errors are reported to the calling function and the connection remains active. (By default, a Callin connection is like execution of a routine in application mode. Any runtime error detected by Caché results in closing the connection and returning error CACHE_CONBROKEN for both the current operation and any subsequent attempts to use Callin without establishing a new connection.)
- CACHE_TTALL — Default. Caché should initialize the terminal's settings and restore them across each call into, and return from, the interface.
- CACHE_TTCALLIN — Caché should initialize the terminal each time it is called but should restore it only when **CacheEnd** is called or the connection is broken.

- CACHE_TTSTART — Caché should initialize the terminal when the connection is formed and reset it when the connection is terminated.
- CACHE_TTNEVER — Caché should not alter the terminal's settings.
- CACHE_TTNONE — Caché should not do any output or input from the principal input/output devices. This is equivalent to specifying the null device for principal input and principal output. **Read** commands from principal input generate an <ENDOFFILE> error and **Write** command to principal output are ignored.
- CACHE_TTNOUSE — This flag is allowed with CACHE_TTALL, CACHE_TTCALLIN, and CACHE_TTSTART. It is implicitly set by the flags CACHE_TTNEVER and CACHE_TTNONE. It indicates that Caché **Open** and **Use** commands are not allowed to alter the terminal, subsequent to the initial open of principal input and principal output.

Return Values for CacheStartW

CACHE_ALREADYCON	Connection already existed. Returned if you call **CacheStartW** from a **$ZF** function.
CACHE_CONBROKEN	Connection was formed and then broken, and **CacheEndW** has not been called to clean up.
CACHE_FAILURE	An unexpected error has occurred.
CACHE_STRTOOLONG	*prinp* or *prout* is too long.
CACHE_SUCCESS	Connection formed.

The flags parameter(s) convey information about how your C program will behave and how you want Caché to set terminal characteristics. The safest, but slowest, route is to have Caché set and restore terminal settings for each call into ObjectScript. However, you can save ObjectScript overhead by handling more of that yourself, and collecting only information that matters to your program. The parameter CACHE_TTNEVER requires the least overhead.

Example

A Caché process is started. The terminal is reset after each interface Callin function. The start fails if a partition is not allocated within 20 seconds. The file dobackup is used for input. It contains an ObjectScript script for a Caché backup. Output appears on the terminal.

```
inpdev;
outdev;
int rc;

strcpy(inpdev.str, "[BATCHDIR]dobackup");
inpdev.len = strlen(inpdev.str);
strcpy(outdev.str,"");
outdev.len = strlen(outdev.str);
rc = CacheStartW(CACHE_TTALL|CACHE_TTNOUSE,0,inpdev,outdev);
```

4.144 CacheTCommit

```
int CacheTCommit( )
```

Description

Executes a Cache TCommit command.

Return Values for CacheTCommit

CACHE_SUCCESS	TCommit was successful.

4.145 CacheTLevel

```
int CacheTLevel( )
```

Description

Returns the current nesting level ($TLEVEL) for transaction processing.

Return Values for CacheTLevel

CACHE_SUCCESS	TLevel was successful.

4.146 CacheTRollback

```
int CacheTRollback(int nlev)
```

Arguments

nlev	Determines how many levels to roll back, (all levels if 0, one level if 1).

Description

Executes a Cache TRollback command. If *nlev* is 0, rolls back all transactions in progress (no matter how many levels of TSTART were issued) and resets $TLEVEL to 0. If *nlev* is 1, rolls back the current level of nested transactions (the one initiated by the most recent TSTART) and decrements $TLEVEL by 1.

Return Values for CacheTRollback

CACHE_SUCCESS	TStart was successful.

4.147 CacheTStart

```
int CacheTStart( )
```

Description

Executes a Cache TStart command.

Return Values for CacheTStart

CACHE_SUCCESS	TStart was successful.

4.148 CacheType

```
int CacheType( )
```

Description

Returns the native type of the item returned by **CacheEvalA**, **CacheEvalW**, or **CacheEvalH** as the function value.

Return Values for CacheType

CACHE_ASTRING	8-bit string.
CACHE_CONBROKEN	Connection has been closed due to a serious error condition or **RESJOB**.
CACHE_DOUBLE	64-bit Caché floating point.
CACHE_ERSYSTEM	Either ObjectScript generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
CACHE_IEEE_DBL	64-bit IEEE floating point.
CACHE_INT	32-bit integer.
CACHE_NOCON	No connection has been established.
CACHE_NORES	No result whose type can be returned (no call to **CacheEvalA** or **CacheEvalW** preceded this call).
CACHE_OREF	Caché object reference.
CACHE_WSTRING	Unicode string.

Example

```
rc = CacheType();
```

4.149 CacheUnPop

```
int CacheUnPop( )
```

Description

Restores the stack entry from **CachePop**.

Return Values for CacheUnPop

CACHE_NORES	No result whose type can be returned has preceded this call.
CACHE_SUCCESS	The operation was successful.