InterSystems™
IRIS Data Platform

Using the Callin API

Version 2019.2
2019-10-10

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

Using the Callin API
InterSystems IRIS Data Platform   Version 2019.2   2019-10-10
Copyright © 2019 InterSystems Corporation
All rights reserved.

Table of Contents

List of Tables

About This Book

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

This book is organized as follows:

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

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 InterSystems IRIS language bindings. For details, see the following sources:

- "InterSystems Java Connectivity Options" in *Using Java with the InterSystems JDBC Driver*
- *Using the InterSystems Managed Provider for .NET*

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

- *Using the Callout Gateway*

1 The Callin Interface

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

- The iris-callin.h Header File
- 8-bit and Unicode String Handling
- Using InterSystems 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 an InterSystems IRIS database. Although InterSystems IRIS currently supports only C and C++ programs, any language that uses the calling standard for that platform 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.

1.1 The iris-callin.h Header File

The iris-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 "iris-callin.h"
```

The iris-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 InterSystems IRIS behaves.

You can translate the distributed header file, iris-callin.h. However, iris-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 `IRIS_SUCCESS`, an InterSystems IRIS error, or a Callin interface error.

There are two types of errors:

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

iris-callin.h defines symbols for all system and interface errors, including `IRIS_SUCCESS` (`0`) and `IRIS_FAILURE` (`-1`). You can translate InterSystems IRIS errors (positive integers) by making a call to the Callin function **IrisErrxlate**.

1.2 8-bit and Unicode String Handling

InterSystems 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,**IrisEvalA** or **IrisPopExStr**) are versions for 8-bit character strings.
- Names with a "W" suffix (for example,**IrisEvalW** or **IrisPopExStrW**) are versions for Unicode character strings on platforms that use 2–byte Unicode characters.
- Names with an "H" suffix (for example,**IrisEvalH** or **IrisPopExStrH**) 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 InterSystems IRIS.

1.2.1 8-bit String Data Types

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

- IRIS_ASTR — counted string of 8-bit characters
- IRIS_ASTRP — Pointer to an 8-bit counted string

The type definition for these is:

```
#define IRIS_MAXSTRLEN 32767
typedef struct {
   unsigned short  len;
   Callin_char_t   str[IRIS_MAXSTRLEN];
} IRIS_ASTR, *IRIS_ASTRP;
```

The `IRIS_ASTR` and `IRIS_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.

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

1.2.2 2–byte Unicode Data Types

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

- IRISWSTR — Unicode counted string
- IRISWSTRP — Pointer to Unicode counted string

The type definition for these is:

```
typedef struct {
   unsigned short len;
   unsigned short str[IRIS_MAXSTRLEN];
} IRISWSTR, *IRISWSTRP;
```

The `IRISWSTR` and `IRISWSTRP` 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.

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

On Unicode-enabled versions of InterSystems IRIS, there is also the data type IRIS_WSTRING, which represents the native string type on 2–byte platforms. **IrisType** returns this type. Also, **IrisConvert** can specify IRIS_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 IRISWSTR buffer.

1.2.3 4–byte Unicode Data Types

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

- IRISHSTR — Extended Unicode counted string
- IRISHSTRP — Pointer to Extended Unicode counted string

The type definition for these is:

```
typedef struct {
   unsigned int len;
   wchar_t str[IRIS_MAXSTRLEN];
} IRISHSTR, *IRISHSTRP;
```

The `IRISHSTR` and `IRISHSTRP` 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.

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

On Unicode-enabled versions of InterSystems IRIS, there is also the data type IRIS_HSTRING, which represents the native string type on 4–byte platforms. **IrisType** returns this type. Also, **IrisConvert** can specify IRIS_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 IRISHSTR buffer.

1.2.4 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 `IRISSTR`, `IRIS_STR`, `IRISSTRP`, or `IRIS_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 `IRIS_UNICODE` and `IRIS_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 iris-callin.h illustrates the concept:

```
#if defined(IRIS_UNICODE) /* Unicode character strings */
#define   IRISSTR      IRISWSTR
#define   IRIS_STR     IRISWSTR
#define   IRISSTRP     IRISWSTRP
#define   IRIS_STRP    IRISWSTRP
#define   IRIS_STRING  IRIS_WSTRING

#elif defined(IRIS_WCHART)  /* wchar_t character strings */
#define   IRISSTR      IRISHSTR
#define   IRIS_STR     IRISHSTR
#define   IRISSTRP     IRISHSTRP
#define   IRIS_STRP    IRISHSTRP
#define   IRIS_STRING  IRIS_HSTRING

#else                  /* 8-bit character strings */
#define   IRISSTR      IRIS_ASTR
#define   IRIS_STR     IRIS_ASTR
#define   IRISSTRP     IRIS_ASTRP
#define   IRIS_STRP    IRIS_ASTRP
#define   IRIS_STRING  IRIS_ASTRING
#endif
```

1.3 Using InterSystems Security Functions

Two functions are provided for working with InterSystems IRIS passwords:

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

There are **IrisSecureStart** and **IrisChangePassword** 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 **IrisStart** entry point.

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

1.4 Using Callin with Multithreading

InterSystems IRIS 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 InterSystems IRIS by calling **IrisSecureStart**. Threads may not share a single connection to

InterSystems IRIS; each thread which wants to use InterSystems IRIS must call **IrisSecureStart**. If a thread attempts to use a Callin function and it has not called **IrisSecureStart**, a `IRIS_NOCON` error is returned.

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

If **IrisSecureStart** is being used, to specify credentials as part of the login, each thread must call **IrisSecureStart** and provide the correct username/password for the connection, since credentials are not shared between the threads. There is a performance penalty within InterSystems IRIS 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.

1.4.1 Threads and UNIX® Signal Handling

On UNIX®, InterSystems IRIS uses a number of signals. If your application uses the same signals, you should be aware of how InterSystems IRIS 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.

1.4.1.1 Signal Processing

InterSystems IRIS integrates with application signal handling by saving application handlers and signal masks, then restoring them at the appropriate time. Signals are processed in the following ways:

Generated signals

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

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

Synchronous Signals

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

Asynchronous Signals

InterSystems IRIS 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, InterSystems IRIS 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.

1.4.1.2 Synchronous Signals

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

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`)

1.4.1.3 Asynchronous signals

Asynchronous signals are generated outside the application (for example, `SIGALRM`, `SIGINT`, and `SIGTERM`). InterSystems IRIS 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 **IrisStart**. On the first call to **IrisStart**, the following actions are performed for all asynchronous signals that would terminate the process:

- InterSystems IRIS looks for a handler for these signals. If a handler is found, InterSystems IRIS leaves it in place. Otherwise, it sets the signal to `SIG_IGN` (ignore the signal).

- InterSystems IRIS 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 InterSystems IRIS 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, InterSystems IRIS will handle it. On receipt of a `SIGTERM` signal, the InterSystems IRIS 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`	InterSystems IRIS asynchronous thread rundown

1.5 Callin Programming Tips

Topics in this section include:

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

1.5.1 Tips for All Callin Programs

Your external program must follow certain rules to avoid corrupting InterSystems IRIS 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 InterSystems IRIS from opening the number of databases or other files it expects to be able to. Normally, InterSystems IRIS 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, InterSystems IRIS expects to have between 6 and 30 InterSystems IRIS 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:\IrisApps\Accounting\AccountsPayable\` directory, this has 40 characters in it and is therefore valid.

- *Call IrisEnd after IrisStart before halting*

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

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

  IrisEnd performs cleanup operations which are necessary to prepare for another call to **IrisStart**. Calling **IrisStart** again without calling **IrisEnd** (assuming a broken connection) will return the code IRIS_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 **IrisContext** to determine whether you are within ObjectScript. This call is particularly important in exit handlers and **Ctrl-C** or **Ctrl-Y** handlers. If **IrisContext** returns a non-zero value, you can invoke **IrisAbort**.

- *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 InterSystems IRIS" in the function descriptions.

- *Avoid signal handling when using IrisStart()*

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

1.5.2 Tips for Windows

These tips apply only to Windows.

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

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

  ```
  <InterSystems IRIS 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 InterSystems IRIS shared memory segment.

1.5.3 Tips for UNIX® and Linux

These tips apply only to UNIX® and Linux.

- *Do not disable interrupt delivery on UNIX®*

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

- *Avoid using reserved signals*

  On UNIX®, InterSystems IRIS uses a number of signals. If possible, application programs linked with InterSystems IRIS 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 InterSystems IRIS deals with them. See Threads and UNIX® Signal Handling for details.

1.6 Running Sample Programs on Windows

The \dev\iris\callin directory contains source files, header files, and project directories for building InterSystems Callin applications. These projects provide a simple demonstration of how to use some high level InterSystems IRIS 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 InterSystems IRIS mgr directory. For a default installation, the shdir.c file will look like this:

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

The Callin interface provides the IRISSETDIR entry point to dynamically set the name of the manager directory at runtime. The shared library version of iris 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 irisdb.obj.
- callinsh — builds a dynamically linked Callin application using irisdb.dll.

- callint — builds a dynamically linked thread-safe Callin application, using irist.dll.

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

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

1.7 Running Sample Programs on UNIX® and Linux

The directory dev/iris/callin/sample contains a complete Makefile to build Callin sample applications.

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

Run make in the dev/iris/callin/samples directory. The supplied Makefile will build a version of iris 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 IRISSETDIR entry point to dynamically set the name of the manager directory at runtime.

Using Makefiles on UNIX®

The UNIX® Makefiles for building Callin sample demos 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 sample 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®

InterSystems IRIS 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 irisusr (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 InterSystems IRIS results in protection errors from the operating system (usually specifying that access is denied); this occurs prior to establishing any connection with InterSystems IRIS.

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

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

2

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 InterSystems IRIS™ 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 InterSystems IRIS 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.

2.1 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.

2.1.1 Session Control

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

Table 2–1: Session control functions

IrisAbort	Tells InterSystems IRIS to terminate the current request.
IrisChangePasswordA[**W**][**H**]	Changes the user's password if InterSystems authentication is used. Must be called before a Callin session is initialized.
IrisContext	Returns an integer indicating whether you are in a **$ZF** callback session, in the InterSystems IRIS side of a Callin call, or in the user program side.
IrisCtrl	Determines whether or not InterSystems IRIS ignores **CTRL-C**.
IrisEnd	Terminates an InterSystems IRIS session and, if necessary, cleans up a broken connection. (Calls into InterSystems IRIS).
IrisEndAll	Disconnects all Callin threads and waits until they terminate.
IrisOflush	Flushes any pending output.
IrisPromptA[**W**][**H**]	Returns a string that would be the Terminal.
IrisSetDir	Dynamically sets the name of the manager's directory (IrisSys\Mgr) at runtime. On Windows, the shared library version of InterSystems IRIS requires this function.
IrisSignal	Reports a signal detected by the user program to InterSystems IRIS for handling.
IrisSecureStartA[**W**][**H**]	Initiates an InterSystems IRIS process.
IrisStartA[**W**][**H**]	(Deprecated. Use **IrisSecureStart** instead) Initiates an InterSystems IRIS process.

2.1.2 Running ObjectScript

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

Table 2–2: ObjectScript command functions

IrisExecuteA[**W**][**H**]	Executes an ObjectScript command. (Calls into InterSystems IRIS).
IrisEvalA[**W**][**H**]	Evaluates an ObjectScript expression. (Calls into InterSystems IRIS).
IrisConvert	Returns the value of the InterSystems IRIS expression returned by **IrisEval**.
IrisType	Returns the datatype of an item returned by **IrisEval**.
IrisErrorA[**W**][**H**]	Returns the most recent error message, its associated source string, and the offset to where in the source string the error occurred.
IrisErrxlateA[**W**][**H**]	Returns the InterSystems IRIS error string associated with error number returned from a Callin function.

2.2 Functions and Routines

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

Table 2–3: Functions for performing function and routine calls

IrisDoFun	Perform a routine call (special case). (Calls into InterSystems IRIS).
IrisDoRtn	Perform a routine call. (Calls into InterSystems IRIS).
IrisExtFun	Perform an extrinsic function call. (Calls into InterSystems IRIS).
IrisPop	Pops a value off argument stack.
IrisUnPop	Restores the stack entry from **IrisPop**
IrisPushFunc[**W**][**H**]	Pushes an extrinsic function reference onto the argument stack.
IrisPushFuncX[**W**][**H**]	Push an extended function reference onto argument stack
IrisPushRtn[**W**][**H**]	Push a routine reference onto argument stack
IrisPushRtnX[**W**][**H**]	Push an extended routine reference onto argument stack

2.3 Transactions and Locking

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

2.3.1 Transactions

The following functions execute the standard InterSystems IRIS transaction commands.

Table 2–4: Transaction functions

IrisTCommit	Executes a TCommit command.
IrisTLevel	Returns the current nesting level ($TLEVEL) for transaction processing.
IrisTRollback	Executes a TRollback command.
IrisTStart	Executes a TStart command.

2.3.2 Locking

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

Table 2–5: Locking functions

IrisAcquireLock	Executes a LOCK command.
IrisReleaseAllLocks	Performs an argumentless InterSystems IRIS LOCK command to remove all locks currently held by the process.
IrisReleaseLock	Executes an InterSystems IRIS LOCK — command to decrement the lock count for the specified lock name.
IrisPushLock[**W**][**H**]	Initializes a IrisAcquireLock command by pushing the lock name on the argument stack.
IrisPushLockX[**W**][**H**]	Initializes a IrisAcquireLock command by pushing the lock name and an environment string on the argument stack.

2.4 Managing Objects

These functions call into InterSystems IRIS 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.

2.4.1 Orefs

Table 2–6: Oref functions

IrisCloseOref	Decrement the reference counter for an OREF. (Calls into InterSystems IRIS).
IrisIncrementCountOref	Increment the reference counter for an OREF
IrisPopOref	Pop an OREF off argument stack
IrisPushOref	Push an OREF onto argument stack

2.4.2 Methods

Table 2–7: Method functions

IrisInvokeMethod	Perform an instance method call. (Calls into InterSystems IRIS).
IrisPushMethod[**W**][**H**]	Push an instance method reference onto argument stack
IrisInvokeClassMethod	Perform a class method call. (Calls into InterSystems IRIS).
IrisPushClassMethod[**W**][**H**]	Push a class method reference onto argument stack

2.4.3 Properties

Table 2–8: Property functions

IrisGetProperty	Obtain the value for a property. (Calls into InterSystems IRIS).
IrisSetProperty	Store the value for a property. (Calls into InterSystems IRIS).
IrisPushProperty[**W**][**H**]	Push a property name onto argument stack

2.5 Managing Globals

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

Table 2–9: Functions for managing globals

IrisGlobalGet	Obtains the value of the global reference defined by **IrisPushGlobal**[**W**][**H**] and any subscripts. The node value is pushed onto the argument stack.
IrisGlobalGetBinary	Obtains the value of the global reference like **IrisGlobalGet**, and also tests to make sure that the result is a binary string that will fit in the provided buffer.
IrisGlobalSet	Stores the value of the global reference. The node value must be pushed onto the argument stack before this call.
IrisGlobalData	Performs a $Data on the specified global.
IrisGlobalIncrement	Performs a $Increment and returns the result on top of the stack.
IrisGlobalKill	Performs a ZKILL on a global node or tree.
IrisGlobalOrder	Performs a $Order on the specified global.
IrisGlobalQuery	Performs a $Query on the specified global.
IrisGlobalRelease	Releases ownership of a retained global buffer, if one exists.
IrisPushGlobal[**W**][**H**]	Pushes a global name onto argument stack
IrisPushGlobalX[**W**][**H**]	Pushes an extended global name onto argument stack

2.6 Managing Strings

These functions translate strings from one form to another, and push or pop string arguments. These string functions may be used for both standard strings and legacy short strings. Functions are provided for local 8-bit encoding, 2–byte Unicode, and 4–byte Unicode.

Table 2–10: String functions

IrisCvtExStrInA[**W**][**H**]	Translates a string with specified external character set encoding to the character string encoding used internally by InterSystems IRIS.
IrisCvtExStrOutA[**W**][**H**]	Translates a string from the character string encoding used internally in InterSystems IRIS to a string with the specified external character set encoding.
IrisExStrKill	Releases the storage associated with a string.
IrisExStrNew[**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.
IrisPopExStr[**W**][**H**]	Pops a value off argument stack and converts it to a string of the desired type.
IrisPushExStr[**W**][**H**]	Pushes a string onto the argument stack

2.7 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 2–11: Other datatype functions

IrisPushInt	Push an integer onto argument stack
IrisPopInt	Pop a value off argument stack and convert it to an integer
IrisPushInt64	Push a 64–bit (long long) value onto argument stack
IrisPopInt64	Pop a value off argument stack and convert it to a 64–bit (long long) value
IrisPushDbl	Push a double onto argument stack
IrisPushIEEEDbl	Push an IEEE double onto argument stack.
IrisPopDbl	Pops value off argument stack and converts it to a double
IrisPushList	Translates and pushes a **$LIST** object onto argument stack
IrisPopList	Pops a **$LIST** object off argument stack and translates it
IrisPushPtr	Pushes a pointer value onto argument stack
IrisPopPtr	Pops a pointer value off argument stack
IrisPushUndef	Pushes an Undefined value that is interpreted as an omitted function argument.
IrisBitFind[**B**]	Returns the position of specified bit values within a bitstring. Similar to InterSystems IRIS $BITFIND.

3 Callin Function Reference

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

Note: InterSystems 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,**IrisEvalA** or **IrisPopExStr**) are versions for 8-bit character strings.
- Names with a “W” suffix (for example,**IrisEvalW** or **IrisPopExStrW**) are versions for Unicode character strings on platforms that use 2–byte Unicode characters.
- Names with an “H” suffix (for example,**IrisEvalH** or **IrisPopExStrH**) 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, **IrisEvalA**[**W**][**H**] or **IrisPopExStr**[**W**][**H**] .

3.1 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.

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

- **IrisCtrl** — Determines whether or not InterSystems IRIS ignores **CTRL-C**.
- **IrisDoFun** — Performs a routine call (special case).
- **IrisDoRtn** — Performs a routine call.
- **IrisEnd** — Terminates an InterSystems IRIS process. If there is a broken connection, it also performs clean-up operations.
- **IrisEndAll** — Disconnects all Callin threads and waits until they terminate.
- **IrisErrorA**[**W**][**H**] — Returns the most recent error message, its associated source string, and the offset to where in the source string the error occurred.
- **IrisErrxlateA**[**W**][**H**] — Translates an integer error code into an InterSystems IRIS error string.
- **IrisEvalA**[**W**][**H**] — Evaluates a string as if it were an InterSystems IRIS expression and places the return value in memory for further processing by **IrisType** and **IrisConvert**.
- **IrisExecuteA**[**W**][**H**] — Executes a command string as if it were typed in the Terminal.
- **IrisExStrKill** — Releases the storage associated with an EXSTR string.
- **IrisExStrNew**[**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.
- **IrisExtFun** — Performs an extrinsic function call where the return value is pushed onto the argument stack.
- **IrisGetProperty** — Obtains the value of the property defined by **IrisPushProperty**[**W**][**H**]. The value is pushed onto the argument stack.
- **IrisGlobalData** — Performs a $Data on the specified global.
- **IrisGlobalGet** — Obtains the value of the global reference defined by **IrisPushGlobal**[**W**][**H**] and any subscripts. The node value is pushed onto the argument stack.
- **IrisGlobalIncrement** — Performs a $INCREMENT and returns the result on top of the stack.
- **IrisGlobalKill** — Performs a ZKILL on a global node or tree.
- **IrisGlobalOrder** — Performs a $Order on the specified global.
- **IrisGlobalQuery** — Performs a $Query on the specified global.
- **IrisGlobalRelease** — Release ownership of a retained global buffer, if one exists.
- **IrisGlobalSet** — Stores the value of the global reference defined by **IrisPushGlobal**[**W**][**H**] and any subscripts. The node value must be pushed onto the argument stack before this call.
- **IrisIncrementCountOref** — Increments the system reference counter for an OREF.
- **IrisInvokeClassMethod** — Executes the class method call defined by **IrisPushClassMethod**[**W**][**H**] and any arguments. The return value is pushed onto the argument stack.
- **IrisInvokeMethod** — Executes the instance method call defined by **IrisPushMethod**[**W**][**H**] and any arguments pushed onto the argument stack.
- **IrisOflush** — Flushes any pending output.
- **IrisPop** — Pops a value off argument stack.
- **IrisPopDbl** — Pops a value off argument stack and converts it to a double.
- **IrisPopExStr**[**W**][**H**] — Pops a value off argument stack and converts it to a string.
- **IrisPopInt** — Pops a value off argument stack and converts it to an integer.
- **IrisPopInt64** — Pops a value off argument stack and converts it to a 64-bit (long long) number.

- **IrisPopList** — Pops a $LIST object off argument stack and converts it.
- **IrisPopOref** — Pops an OREF off argument stack.
- **IrisPopPtr** — Pops a pointer off argument stack in internal format.
- **IrisPromptA**[**W**][**H**] — Returns a string that would be the Terminal.
- **IrisPushClassMethod**[**W**][**H**] — Pushes a class method reference onto the argument stack.
- **IrisPushDbl** — Pushes a double onto the argument stack.
- **IrisPushExStr**[**W**][**H**] — Pushes a string onto the argument stack.
- **IrisPushFunc**[**W**][**H**] — Pushes an extrinsic function reference onto the argument stack.
- **IrisPushFuncX**[**W**][**H**] — Pushes an extended extrinsic function reference onto the argument stack.
- **IrisPushGlobal**[**W**][**H**] — Pushes a global reference onto the argument stack.
- **IrisPushGlobalX**[**W**][**H**] — Pushes an extended global reference onto the argument stack.
- **IrisPushIEEEDbl** — Pushes an IEEE double onto the argument stack.
- **IrisPushInt** — Pushes an integer onto the argument stack.
- **IrisPushInt64** — Pushes a 64-bit (long long) number onto the argument stack.
- **IrisPushList** — Converts a $LIST object and pushes it onto the argument stack.
- **IrisPushLock**[**W**][**H**] — Initializes a **IrisAcquireLock** command by pushing the lock name on the argument stack.
- **IrisPushLockX**[**W**][**H**] — Initializes a **IrisAcquireLock** command by pushing the lock name and an environment string on the argument stack.
- **IrisPushMethod**[**W**][**H**] — Pushes an instance method reference onto the argument stack.
- **IrisPushOref** — Pushes an OREF onto the argument stack.
- **IrisPushProperty**[**W**][**H**] — Pushes a property reference onto the argument stack.
- **IrisPushPtr** — Pushes a pointer onto the argument stack in internal format.
- **IrisPushRtn**[**W**][**H**] — Pushes a routine reference onto the argument stack.
- **IrisPushRtnX**[**W**][**H**] — Pushes an extended routine reference onto the argument stack.
- **IrisPushUndef** — pushes an Undefined value that is interpreted as an omitted function argument.
- **IrisReleaseAllLocks** — Performs an argumentless InterSystems IRIS LOCK command to remove all locks currently held by the process.
- **IrisReleaseLock** — Executes an InterSystems IRIS LOCK command to decrement the lock count for the specified lock name. This command will only release one incremental lock at a time.
- **IrisSecureStartA**[**W**][**H**] — Calls into InterSystems IRIS to set up a process.
- **IrisSetDir** — Dynamically sets the name of the manager's directory at runtime.
- **IrisSetProperty** — Stores the value of the property defined by **IrisPushProperty**[**W**][**H**].
- **IrisSignal** — Passes on signals caught by user's program to InterSystems IRIS.
- **IrisSPCReceive** — Receive single-process-communication message.
- **IrisSPCSend** — Send a single-process-communication message.
- **IrisStartA**[**W**][**H**] — Calls into InterSystems IRIS to set up an InterSystems IRIS process.
- **IrisTCommit** — Executes an InterSystems IRIS TCommit command.

- **IrisTLevel** — Returns the current nesting level ($TLEVEL) for transaction processing.
- **IrisTRollback** — Executes an InterSystems IRIS TRollback command.
- **IrisTStart** — Executes an InterSystems IRIS TStart command.
- **IrisType** — Returns the native type of the item returned by **IrisEvalA**[**W**][**H**], as the function value.
- **IrisUnPop** — Restores the stack entry from **IrisPop**.

3.2 IrisAbort

```
int IrisAbort(unsigned long type)
```

Arguments

type	Either of the following predefined values that specify how the termination occurs: • `IRIS_CTRLC` — Interrupts the InterSystems IRIS processing as if a **CTRL-C** had been processed (regardless of whether **CTRL-C** has been enabled with **IrisCtrl**). A connection to InterSystems IRIS remains. • `IRIS_RESJOB` — Terminates the Callin connection. You must then call **IrisEnd** and then **IrisStart** to reconnect to InterSystems IRIS.

Description

Tells InterSystems IRIS to cancel the current request being processed on the InterSystems IRIS 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 **IrisContext** to determine if there is an InterSystems IRIS request currently being processed.) Note that this only applies to Callin programs that use an AST or separate thread.

Return Values for IrisAbort

IRIS_BADARG	The termination type is invalid.
IRIS_CONBROKEN	Connection has been broken.
IRIS_NOCON	No connection has been established.
IRIS_NOTINCACHE	The Callin partner is not in InterSystems IRIS at this time.
IRIS_SUCCESS	Connection formed.

Example

```
rc = IrisAbort(IRIS_CTRLC);
```

3.3 IrisAcquireLock

```
int IrisAcquireLock(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 IRIS_INCREMENTAL_LOCK and IRIS_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 an InterSystems IRIS LOCK command. The lock reference should already be set up with **IrisPushLock**.

Return Values for IrisAcquireLock

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

3.4 IrisBitFind

```
int IrisBitFind(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 InterSystems IRIS $BITFIND (also see "General Information on Bitstring Functions").

Return Values for IrisBitFind

IRIS_SUCCESS	The operation was successful.

3.5 IrisBitFindB

```
int IrisBitFindB(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 InterSystems IRIS $BITFIND (also see "General Information on Bitstring Functions").

Return Values for IrisBitFindB

IRIS_SUCCESS	The operation was successful.

3.6 IrisChangePasswordA

Variants: **IrisChangePasswordW**, **IrisChangePasswordH**

```
int IrisChangePasswordA(IRIS_ASTRP username, IRIS_ASTRP oldpassword, IRIS_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 InterSystems authentication is used. It is not valid for LDAP, DELEGATED, Kerberos, or other forms of authentication. It must be called before a Callin session is initialized. A typical use would be to handle a `IRIS_CHANGEPASSWORD` error from **IrisSecureStart**. In such a case **IrisChangePassword** would be called to change the password, then **IrisSecureStart** would be called again.

Return Values for IrisChangePasswordA

IRIS_FAILURE	An unexpected error has occurred.
IRIS_SUCCESS	Password changed.

3.7 IrisChangePasswordH

Variants: **IrisChangePasswordA**, **IrisChangePasswordW**

```
int IrisChangePasswordH(IRISHSTRP username, IRISHSTRP oldpassword, IRISHSTRP 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 InterSystems authentication is used. It is not valid for LDAP, DELEGATED, Kerberos, or other forms of authentication. It must be called before a Callin session is initialized. A typical use would be to handle a `IRIS_CHANGEPASSWORD` error from **IrisSecureStart**. In such a case **IrisChangePassword** would be called to change the password, then **IrisSecureStart** would be called again.

Return Values for IrisChangePasswordH

IRIS_FAILURE	An unexpected error has occurred.
IRIS_SUCCESS	Password changed.

3.8 IrisChangePasswordW

Variants: **IrisChangePasswordA**, **IrisChangePasswordH**

```
int IrisChangePasswordW(IRISWSTRP username, IRISWSTRP oldpassword, IRISWSTRP 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 InterSystems authentication is used. It is not valid for LDAP, DELEGATED, Kerberos, or other forms of authentication. It must be called before a Callin session is initialized. A typical use would be to handle a `IRIS_CHANGEPASSWORD` error from **IrisSecureStart**. In such a case **IrisChangePassword** would be called to change the password, then **IrisSecureStart** would be called again.

Return Values for IrisChangePasswordW

IRIS_FAILURE	An unexpected error has occurred.
IRIS_SUCCESS	Password changed.

3.9 IrisCloseOref

```
int IrisCloseOref(unsigned int oref)
```

Arguments

oref	Object reference.

Description

Decrements the system reference counter for an OREF.

Return Values for IrisCloseOref

IRIS_ERBADOREF	Invalid OREF.
IRIS_SUCCESS	The operation was successful.

3.10 IrisContext

```
int IrisContext()
```

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 **IrisContext** tells you if there is a request currently being processed on the InterSystems IRIS side of the connection. This information is needed to decide if you must return to InterSystems IRIS to allow processing to complete.

Return Values for IrisContext

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

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 InterSystems IRIS to allow processing to complete.

Example

```
rc = IrisContext();
```

3.11 IrisConvert

```
int IrisConvert(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 IRIS_ASTRING, *rbuf* should be the address of a IRIS_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 IRIS_WSTRING, *rbuf* should be the address of a IRISWSTR structure whose *len* element has been filled in to represent the maximum size (in characters).

Description

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

Valid values of *type* are:

- IRIS_ASTRING — 8-bit character string.
- IRIS_CHAR — 8-bit signed integer.
- IRIS_DOUBLE — 64-bit floating point.
- IRIS_FLOAT — 32-bit floating point.
- IRIS_INT — 32-bit signed integer.
- IRIS_INT2 — 16-bit signed integer.
- IRIS_INT4 — 32-bit signed integer.
- IRIS_INT8 — 64-bit signed integer.
- IRIS_UCHAR — 8-bit unsigned integer.
- IRIS_UINT — 32-bit unsigned integer.
- IRIS_UINT2 — 16-bit unsigned integer.
- IRIS_UINT4 — 32-bit unsigned integer.
- IRIS_UINT8 — 64-bit unsigned integer.
- IRIS_WSTRING — Unicode character string.

Return Values for IrisConvert

IRIS_BADARG	Type is invalid.
IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_ERSYSTEM	Either ObjectScript generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
IRIS_FAILURE	An unexpected error has occurred.
IRIS_NOCON	No connection has been established.
IRIS_NORES	No result whose type can be returned (no call to **IrisEvalA** preceded this call).
IRIS_RETTRUNC	Success, but the type IRIS_ASTRING, IRIS_INT8, IRIS_UINT8 and IRIS_WSTRING resulted in a value that would not fit in the space allocated in *retval*. For IRIS_INT8 and IRIS_UINT8, this means that the expression resulted in a floating point number that could not be normalized to fit within 64 bits.
IRIS_STRTOOLONG	String is too long.
IRIS_SUCCESS	Value returned by last **IrisEval** converted successfully.

Note: InterSystems IRIS may perform division when calculating the return value for floating point types, IRIS_FLOAT and IRIS_DOUBLE, which have decimal parts (including negative exponents), as well as the 64-bit integer types (IRIS_INT8 and IRIS_UINT8). Therefore, the returned result may not be identical in value to the original. IRIS_ASTRING, IRIS_INT8, IRIS_UINT8 and IRIS_WSTRING can return the status IRIS_RETTRUNC.

Example

```
IRIS_ASTR retval;
/* define variable retval */

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

rc = IrisConvert(IRIS_ASTRING,&retval);
```

3.12 IrisCtrl

```
int IrisCtrl(unsigned long flags)
```

Arguments

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

Description

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

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

Return Values for IrisCtrl

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

Example

```
rc = IrisCtrl(IRIS_ENABCTRLC);
```

3.13 IrisCvtExStrInA

Variants: **IrisCvtExStrInW**, **IrisCvtExStrInH**

```
int IrisCvtExStrInA(IRIS_EXSTRP src, IRIS_ASTRP tbl, IRIS_EXSTRP res)
```

Arguments

src	Address of a IRIS_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 IRIS_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.

Return Values for IrisCvtExStrInA

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

3.14 IrisCvtExStrInW

Variants: **IrisCvtExStrInA**, **IrisCvtExStrInH**

```
int IrisCvtExStrInW(IRIS_EXSTRP src, IRISWSTRP tbl, IRIS_EXSTRP res)
```

Arguments

src	Address of a IRIS_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 IRIS_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 InterSystems IRIS.

Return Values for IrisCvtExStrInW

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

3.15 IrisCvtExStrInH

Variants: **IrisCvtExStrInA**, **IrisCvtExStrInW**

```
int IrisCvtExStrInH(IRIS_EXSTRP src, IRISWSTRP tbl, IRIS_EXSTRP res)
```

Arguments

src	Address of a IRIS_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 IRIS_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 InterSystems IRIS.

Return Values for IrisCvtExStrInH

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

3.16 IrisCvtExStrOutA

Variants: **IrisCvtExStrOutW**, **IrisCvtExStrOutH**

```
int IrisCvtExStrOutA(IRIS_EXSTRP src, IRIS_ASTRP tbl, IRIS_EXSTRP res)
```

Arguments

src	Address of a IRIS_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 IRIS_EXSTRP variable that will contain the result.

Description

Translates a string from the 8-bit character string encoding used internally in an older InterSystems 8-bit product to a string with the specified external character set encoding.

Return Values for IrisCvtExStrOutA

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

3.17 IrisCvtExStrOutW

Variants: **IrisCvtExStrOutA**, **IrisCvtExStrOutH**

```
int IrisCvtExStrOutW(IRIS_EXSTRP src, IRISWSTRP tbl, IRIS_EXSTRP res)
```

Arguments

src	Address of a IRIS_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 IRIS_EXSTRP variable that will contain the result.

Description

Translates a string from the 2–byte Unicode character string encoding used internally in InterSystems IRIS to a string with the specified external character set encoding.

Return Values for IrisCvtExStrOutW

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

3.18 IrisCvtExStrOutH

Variants: **IrisCvtExStrOutA**, **IrisCvtExStrOutW**

```
int IrisCvtExStrOutH(IRIS_EXSTRP src, IRISWSTRP tbl, IRIS_EXSTRP res)
```

Arguments

src	Address of a IRIS_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 IRIS_EXSTRP variable that will contain the result.

Description

Translates a string from the 4–byte Unicode character string encoding used internally in InterSystems IRIS to a string with the specified external character set encoding.

Return Values for IrisCvtExStrOutH

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

3.19 IrisDoFun

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

Arguments

flags	Routine flags from **IrisPushRtn[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 IrisDoFun

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_FAILURE	Internal consistency error.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.20 IrisDoRtn

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

Arguments

flags	Routine flags from **IrisPushRtn[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 IrisDoRtn

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_FAILURE	Internal consistency error.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.21 IrisEnd

```
int IrisEnd()
```

Description

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

Return Values for IrisEnd

IRIS_FAILURE	Returns if called from a **$ZF** function (rather than from within a Callin executable).
IRIS_NOCON	No connection has been established.
IRIS_SUCCESS	InterSystems IRIS session terminated/cleaned up.

IrisEnd can also return any of the InterSystems IRIS error codes.

Example

```
rc = IrisEnd();
```

3.22 IrisEndAll

```
int IrisEndAll()
```

Description

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

Return Values for IrisEndAll

IRIS_SUCCESS	InterSystems IRIS session terminated/cleaned up.

Example

```
rc = IrisEndAll();
```

3.23 IrisErrorA

Variants: **IrisErrorW**, **IrisErrorH**

```
int IrisErrorA(IRIS_ASTRP msg, IRIS_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 IrisErrorA

IRIS_CONBROKEN	Connection has been broken.
IRIS_NOCON	No connection has been established.
IRIS_RETTOOSMALL	The length of the return value for either *errmsg* or *errsrc* was not of the valid size.
IRIS_SUCCESS	Connection formed.

Example

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

3.24 IrisErrorH

Variants: **IrisErrorA**, **IrisErrorW**

```
int IrisErrorH(IRISHSTRP msg, IRISHSTRP 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 IrisErrorH

IRIS_CONBROKEN	Connection has been broken.
IRIS_NOCON	No connection has been established.
IRIS_RETTOOSMALL	The length of the return value for either *errmsg* or *errsrc* was not of the valid size.
IRIS_SUCCESS	Connection formed.

Example

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

3.25 IrisErrorW

Variants: **IrisErrorA**, **IrisErrorH**

```
int IrisErrorW(IRISWSTRP msg, IRISWSTRP 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 IrisErrorW

IRIS_CONBROKEN	Connection has been broken.
IRIS_NOCON	No connection has been established.
IRIS_RETTOOSMALL	The length of the return value for either *errmsg* or *errsrc* was not of the valid size.
IRIS_SUCCESS	Connection formed.

Example

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

3.26 IrisErrxlateA

Variants: **IrisErrxlateW**, **IrisErrxlateH**

```
int IrisErrxlateA(int code, IRIS_ASTRP rbuf)
```

Arguments

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

Description

Translates error code *code* into an InterSystems IRIS error string, and writes that string into the structure pointed to by *rbuf*

Return Values for IrisErrxlateA

IRIS_ERUNKNOWN	The specified code is less than 1 (in the range of the Callin interface errors) or is above the largest InterSystems IRIS error number.
IRIS_RETTRUNC	The associated error string was truncated to fit in the allocated area.
IRIS_SUCCESS	Connection formed.

Example

```
IRIS_ASTR retval; /* define variable retval */
retval.len = 30; /* maximum return length of string */
rc = IrisErrxlateA(IRIS_ERSTORE,&retval);
```

3.27 IrisErrxlateH

Variants: **IrisErrxlateA**, **IrisErrxlateW**

```
int IrisErrxlateH(int code, IRISHSTRP rbuf)
```

Arguments

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

Description

Translates error code *code* into an InterSystems IRIS error string, and writes that string into the structure pointed to by *rbuf*

Return Values for IrisErrxlateH

IRIS_ERUNKNOWN	The specified code is less than 1 (in the range of the Callin interface errors) or is above the largest InterSystems IRIS error number.
IRIS_RETTRUNC	The associated error string was truncated to fit in the allocated area.
IRIS_SUCCESS	Connection formed.

Example

```
IRISHSTR retval; /* define variable retval */
retval.len = 30; /* maximum return length of string */
rc = IrisErrxlateH(IRIS_ERSTORE,&retval);
```

3.28 IrisErrxlateW

Variants: **IrisErrxlateA**, **IrisErrxlateH**

```
int IrisErrxlateW(int code, IRISWSTRP rbuf)
```

Arguments

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

Description

Translates error code *code* into an InterSystems IRIS error string, and writes that string into the structure pointed to by *rbuf*

Return Values for IrisErrxlateW

IRIS_ERUNKNOWN	The specified code is less than 1 (in the range of the Callin interface errors) or is above the largest InterSystems IRIS error number.
IRIS_RETTRUNC	The associated error string was truncated to fit in the allocated area.
IRIS_SUCCESS	Connection formed.

Example

```
IRISWSTR retval; /* define variable retval */
retval.len = 30; /* maximum return length of string */
rc = IrisErrxlateW(IRIS_ERSTORE,&retval);
```

3.29 IrisEvalA

Variants: **IrisEvalW**, **IrisEvalH**

```
int IrisEvalA(IRIS_ASTRP volatile expr)
```

Arguments

expr	The address of a IRIS_ASTR variable.

Description

Evaluates a string as if it were an InterSystems IRIS expression and places the return value in memory for further processing by **IrisType** and **IrisConvert**.

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

CAUTION: The next call to **IrisEvalA**, **IrisExecuteA**, or **IrisEnd** will overwrite the existing return value.

Return Values for IrisEvalA

IRIS_CONBROKEN	Connection has been closed due to a serious error condition or **RESJOB**.
IRIS_ERSYSTEM	Either InterSystems IRIS generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
IRIS_NOCON	No connection has been established.
IRIS_STRTOOLONG	String is too long.
IRIS_SUCCESS	String evaluated successfully.

IrisEvalA can also return any of the InterSystems IRIS error codes.

Example

```
int rc;
IRIS_ASTR retval;
IRIS_ASTR expr;

strcpy(expr.str, "\"Record\"_^Recnum_\" = \"_$$^GetRec(^Recnum)");
expr.len = strlen(expr.str);
rc = IrisEvalA(&expr);
if (rc == IRIS_SUCCESS)
    rc = IrisConvert(IRIS_ASTRING,&retval);
```

3.30 IrisEvalH

Variants: **IrisEvalA**, **IrisEvalW**

```
int IrisEvalH(IRISHSTRP volatile expr)
```

Arguments

expr	The address of a IRISHSTRP variable.

Description

Evaluates a string as if it were an InterSystems IRIS expression and places the return value in memory for further processing by **IrisType** and **IrisConvert**.

If **IrisEvalH** completes successfully, it sets a flag that allows calls to **IrisType** and **IrisConvert** to complete. These functions are used to process the item returned from **IrisEvalA**.

CAUTION: The next call to **IrisEvalH**, **IrisExecuteH**, or **IrisEnd** will overwrite the existing return value.

Return Values for IrisEvalH

IRIS_CONBROKEN	Connection has been closed due to a serious error condition or **RESJOB**.
IRISW_ERSYSTEM	Either InterSystems IRIS generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
IRIS_NOCON	No connection has been established.
IRIS_STRTOOLONG	String is too long.
IRIS_SUCCESS	String evaluated successfully.

IrisEvalH can also return any of the InterSystems IRIS error codes.

Example

```
int rc;
IRISHSTRP retval;
IRISHSTRP expr;

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

3.31 IrisEvalW

Variants: **IrisEvalA**, **IrisEvalH**

```
int IrisEvalW(IRISWSTRP volatile expr)
```

Arguments

expr	The address of a IRISWSTR variable.

Description

Evaluates a string as if it were an InterSystems IRIS expression and places the return value in memory for further processing by **IrisType** and **IrisConvert**.

If **IrisEvalW** completes successfully, it sets a flag that allows calls to **IrisType** and **IrisConvert** to complete. These functions are used to process the item returned from **IrisEvalA**.

CAUTION: The next call to **IrisEvalW**, **IrisExecuteW**, or **IrisEnd** will overwrite the existing return value.

Return Values for IrisEvalW

IRIS_CONBROKEN	Connection has been closed due to a serious error condition or **RESJOB**.
IRISHW_ERSYSTEM	Either InterSystems IRIS generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
IRIS_NOCON	No connection has been established.
IRIS_STRTOOLONG	String is too long.
IRIS_SUCCESS	String evaluated successfully.

IrisEvalW can also return any of the InterSystems IRIS error codes.

Example

```
int rc;
IRISWSTR retval;
IRISWSTR expr;

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

3.32 IrisExecuteA

Variants: **IrisExecuteW**, **IrisExecuteH**

```
int IrisExecuteA(IRIS_ASTRP volatile cmd)
```

Arguments

cmd	The address of a IRIS_ASTR variable.

Description

Executes the command *string* as if it were typed in the Terminal.

CAUTION: The next call to **IrisEvalA**, **IrisExecuteA**, or **IrisEnd** will overwrite the existing return value.

Return Values for IrisExecuteA

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

IrisExecuteA can also return any of the InterSystems IRIS error codes.

Example

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

3.33 IrisExecuteH

Variants: **IrisExecuteA**, **IrisExecuteW**

```
int IrisExecuteH(IRISHSTRP volatile cmd)
```

Arguments

cmd	The address of a IRIS_ASTR variable.

Description

Executes the command *string* as if it were typed in the Terminal.

If **IrisExecuteH** completes successfully, it sets a flag that allows calls to **IrisType** and **IrisConvert** to complete. These functions are used to process the item returned from **IrisEvalH**.

CAUTION: The next call to **IrisEvalH**, **IrisExecuteH**, or **IrisEnd** will overwrite the existing return value.

Return Values for IrisExecuteH

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

IrisExecuteH can also return any of the InterSystems IRIS error codes.

Example

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

3.34 IrisExecuteW

Variants: **IrisExecuteA**, **IrisExecuteH**

```
int IrisExecuteW(IRISWSTRP volatile cmd)
```

Arguments

cmd	The address of a IRIS_ASTR variable.

Description

Executes the command *string* as if it were typed in the Terminal.

If **IrisExecuteW** completes successfully, it sets a flag that allows calls to **IrisType** and **IrisConvert** to complete. These functions are used to process the item returned from **IrisEvalW**.

CAUTION: The next call to **IrisEvalW**, **IrisExecuteW**, or **IrisEnd** will overwrite the existing return value.

Return Values for IrisExecuteW

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

IrisExecuteW can also return any of the InterSystems IRIS error codes.

Example

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

3.35 IrisExStrKill

```
int IrisExStrKill(IRIS_EXSTRP obj)
```

Arguments

obj	Pointer to the string.

Description

Releases the storage associated with an EXSTR string.

Return Values for IrisExStrKill

IRIS_ERUNIMPLEMENTED	String is undefined.
IRIS_SUCCESS	String storage has been released.

3.36 IrisExStrNew

Variants: **IrisExStrNewW**, **IrisExStrNewH**

```
unsigned char * IrisExStrNew(IRIS_EXSTRP zstr, int size)
```

Arguments

zstr	Pointer to a IRIS_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 IrisExStrNew

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

3.37 IrisExStrNewW

Variants: **IrisExStrNew**, **IrisExStrNewH**

```
unsigned short * IrisExStrNewW(IRIS_EXSTRP zstr, int size)
```

Arguments

zstr	Pointer to a IRIS_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 IrisExStrNewW

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

3.38 IrisExStrNewH

Variants: **IrisExStrNew**, **IrisExStrNewW**

```
unsigned short * IrisExStrNewH(IRIS_EXSTRP zstr, int size)
```

Arguments

zstr	Pointer to a IRIS_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 IrisExStrNewH

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

3.39 IrisExtFun

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

Arguments

flags	Routine flags from **IrisPushFunc[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 IrisExtFun

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_FAILURE	Internal consistency error.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.40 IrisGetProperty

```
int IrisGetProperty()
```

Description

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

Return Values for IrisGetProperty

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.41 IrisGlobalData

```
int IrisGlobalData(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 IrisGlobalData

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_ERPROTECT	Protection violation.
IRIS_ERUNDEF	Node has no associated value.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.42 IrisGlobalGet

```
int IrisGlobalGet(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 IRIS_ERUNDEF • `1` — returns IRIS_SUCCESS but the return value is an empty string.

Description

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

Return Values for IrisGlobalGet

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_ERPROTECT	Protection violation.
IRIS_ERUNDEF	Node has no associated value.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.43 IrisGlobalGetBinary

```
int IrisGlobalGetBinary(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 IRIS_ERUNDEF • `1` — returns IRIS_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 **IrisPushGlobal**[**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 IrisGlobalGetBinary

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_ERPROTECT	Protection violation.
IRIS_ERUNDEF	Node has no associated value.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.44 IrisGlobalIncrement

```
int IrisGlobalIncrement(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 IrisGlobalIncrement

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_ERPROTECT	Protection violation.
IRIS_ERUNDEF	Node has no associated value.
IRIS_ERMAXINCR	MAXINCREMENT system error
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.45 IrisGlobalKill

```
int IrisGlobalKill(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 IrisGlobalKill

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_ERPROTECT	Protection violation.
IRIS_ERUNDEF	Node has no associated value.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.46 IrisGlobalOrder

```
int IrisGlobalOrder(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 IrisGlobalOrder

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_ERPROTECT	Protection violation.
IRIS_ERUNDEF	Node has no associated value.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.47 IrisGlobalQuery

```
int IrisGlobalQuery(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 IrisGlobalQuery

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_ERPROTECT	Protection violation.
IRIS_ERUNDEF	Node has no associated value.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.48 IrisGlobalRelease

```
int IrisGlobalRelease( )
```

Description

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

Return Values for IrisGlobalRelease

IRIS_SUCCESS	The operation was successful.

3.49 IrisGlobalSet

```
int IrisGlobalSet(int narg)
```

Arguments

narg	Number of subscript expressions pushed onto the argument stack.

Description

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

Return Values for IrisGlobalSet

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.50 IrisIncrementCountOref

```
int IrisIncrementCountOref(unsigned int oref)
```

Arguments

oref	Object reference.

Description

Increments the system reference counter for an OREF.

Return Values for IrisIncrementCountOref

IRIS_ERBADOREF	Invalid OREF.
IRIS_SUCCESS	The operation was successful.

3.51 IrisInvokeClassMethod

```
int IrisInvokeClassMethod(int narg)
```

Arguments

narg	Number of call arguments pushed onto the argument stack.

Description

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

Return Values for IrisInvokeClassMethod

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.52 IrisInvokeMethod

```
int IrisInvokeMethod(int narg)
```

Arguments

narg	Number of call arguments pushed onto the argument stack.

Description

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

Return Values for IrisInvokeMethod

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.53 IrisOflush

```
int IrisOflush()
```

Description

Flushes any pending output.

Return Values for IrisOflush

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

3.54 IrisPop

```
int IrisPop(void ** arg)
```

Arguments

arg	Pointer to argument stack entry.

Description

Pops a value off argument stack.

Return Values for IrisPop

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

3.55 IrisPopDbl

```
int IrisPopDbl(double * nump)
```

Arguments

nump	Pointer to double value.

Description

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

Return Values for IrisPopDbl

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

3.56 IrisPopExStr

Variants: **IrisPopExStrW**, **IrisPopExStrH**

```
int IrisPopExStr(IRIS_EXSTRP sstrp)
```

Arguments

sstrp	Pointer to standard string pointer.

Description

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

Return Values for IrisPopExStr

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

3.57 IrisPopExStrW

Variants: **IrisPopExStr**, **IrisPopExStrH**

```
int IrisPopExStrW(IRIS_EXSTRP sstrp)
```

Arguments

sstrp	Pointer to standard string pointer.

Description

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

Return Values for IrisPopExStrW

IRIS_NORES	No result whose type can be returned has preceded this call.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
IRIS_EXSTR_INUSE	Returned if *sstrp* has not been initialized to `NULL`.

3.58 IrisPopExStrH

Variants: **IrisPopExStr**, **IrisPopExStrW**

```
int IrisPopExStrH(IRIS_EXSTRP sstrp)
```

Arguments

sstrp	Pointer to standard string pointer.

Description

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

Return Values for IrisPopExStrH

IRIS_NORES	No result whose type can be returned has preceded this call.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
IRIS_EXSTR_INUSE	Returned if *sstrp* has not been initialized to `NULL`.

3.59 IrisPopInt

```
int IrisPopInt(int* nump)
```

Arguments

nump	Pointer to integer value.

Description

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

Return Values for IrisPopInt

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

3.60 IrisPopInt64

```
int IrisPopInt64(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 IrisPopInt64

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

3.61 IrisPopList

```
int IrisPopList(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.

Return Values for IrisPopList

IRIS_NORES	No result whose type can be returned has preceded this call.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.

3.62 IrisPopOref

```
int IrisPopOref(unsigned int * orefp)
```

Arguments

orefp	Pointer to OREF value.

Description

Pops an OREF off argument stack.

Return Values for IrisPopOref

IRIS_NORES	No result whose type can be returned has preceded this call.
IRIS_ERNOOREF	Result is not an OREF.
IRIS_SUCCESS	The operation was successful.

3.63 IrisPopPtr

```
int IrisPopPtr(void ** ptrp)
```

Arguments

ptrp	Pointer to generic pointer.

Description

Pops a pointer off argument stack in internal format.

Return Values for IrisPopPtr

IRIS_NORES	No result whose type can be returned has preceded this call.
IRIS_BADARG	The entry is not a valid pointer.
IRIS_SUCCESS	The operation was successful.

3.64 IrisPromptA

Variants: **IrisPromptW**, **IrisPromptH**

```
int IrisPromptA(IRIS_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 Terminal (without the “>”).

Return Values for IrisPromptA

IRIS_CONBROKEN	Connection has been broken.
IRIS_ERSYSTEM	Either ObjectScript generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
IRIS_FAILURE	An unexpected error has occurred.
IRIS_NOCON	No connection has been established.
IRIS_RETTOOSMALL	*rbuf* must have a length of at least five.
IRIS_SUCCESS	Connection formed.

Example

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

3.65 IrisPromptH

Variants: **IrisPromptA**, **IrisPromptW**

```
int IrisPromptH(IRISHSTRP rbuf)
```

Arguments

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

Description

Returns a string that would be the Terminal (without the “>”).

Return Values for IrisPromptH

IRIS_CONBROKEN	Connection has been broken.
IRIS_ERSYSTEM	Either ObjectScript generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
IRIS_FAILURE	Request failed.
IRIS_NOCON	No connection has been established.
IRIS_RETTOOSMALL	*rbuf* must have a length of at least five.
IRIS_SUCCESS	Connection formed.

Example

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

3.66 IrisPromptW

Variants: **IrisPromptA**, **IrisPromptH**

```
int IrisPromptW(IRISWSTRP rbuf)
```

Arguments

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

Description

Returns a string that would be the Terminal (without the “>”).

Return Values for IrisPromptW

IRIS_CONBROKEN	Connection has been broken.
IRIS_ERSYSTEM	Either ObjectScript generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
IRIS_FAILURE	Request failed.
IRIS_NOCON	No connection has been established.
IRIS_RETTOOSMALL	*rbuf* must have a length of at least five.
IRIS_SUCCESS	Connection formed.

Example

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

3.67 IrisPushClassMethod

Variants: **IrisPushClassMethodW**, **IrisPushClassMethodH**

```
int IrisPushClassMethod(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 IrisPushClassMethod

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_BADARG	Invalid call argument.
IRIS_SUCCESS	The operation was successful.

3.68 IrisPushClassMethodH

Variants: **IrisPushClassMethod**, **IrisPushClassMethodW**

```
int IrisPushClassMethodH(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 IrisPushClassMethodH

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_BADARG	Invalid call argument.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.69 IrisPushClassMethodW

Variants: **IrisPushClassMethod**, **IrisPushClassMethodH**

```
int IrisPushClassMethodW(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 IrisPushClassMethodW

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_BADARG	Invalid call argument.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.70 IrisPushDbl

```
int IrisPushDbl(double num)
```

Arguments

num	Double value.

Description

Pushes a double onto the argument stack.

Return Values for IrisPushDbl

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_SUCCESS	The operation was successful.

3.71 IrisPushExStr

Variants: **IrisPushExStrW**, **IrisPushExStrH**

```
int IrisPushExStr(IRIS_EXSTRP sptr)
```

Arguments

sptr	Pointer to the argument value.

Description

Pushes a string onto the argument stack.

Return Values for IrisPushExStr

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.

3.72 IrisPushExStrW

Variants: **IrisPushExStr**, **IrisPushExStrH**

```
int IrisPushExStrW(IRIS_EXSTRP sptr)
```

Arguments

sptr	Pointer to the argument value.

Description

Pushes a Unicode string onto the argument stack.

Return Values for IrisPushExStrW

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.

3.73 IrisPushExStrH

Variants: **IrisPushExStr**, **IrisPushExStrW**

```
int IrisPushExStrH(IRIS_EXSTRP sptr)
```

Arguments

sptr	Pointer to the argument value.

Description

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

Return Values for IrisPushExStrH

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.

3.74 IrisPushFunc

Variants: **IrisPushFuncW**, **IrisPushFuncH**

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

Arguments

rflag	Routine flags for use by **IrisExtFun**.
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 IrisPushFunc

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.

3.75 IrisPushFuncH

Variants: **IrisPushFunc**, **IrisPushFuncW**

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

Arguments

rflag	Routine flags for use by **IrisExtFun**.
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 IrisPushFuncH

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.76 IrisPushFuncW

Variants: **IrisPushFunc**, **IrisPushFuncH**

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

Arguments

rflag	Routine flags for use by **IrisExtFun**.
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 IrisPushFuncW

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.77 IrisPushFuncX

Variants: **IrisPushFuncXW**, **IrisPushFuncXH**

```
int IrisPushFuncX(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 **IrisExtFun**.
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 IrisPushFuncX

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.

3.78 IrisPushFuncXH

Variants: **IrisPushFuncX**, **IrisPushFuncXW**

```
int IrisPushFuncXH(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 **IrisExtFun**.
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 IrisPushFuncXH

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.79 IrisPushFuncXW

Variants: **IrisPushFuncX**, **IrisPushFuncXH**

```
int IrisPushFuncXW(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 **IrisExtFun**.
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 IrisPushFuncXW

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.80 IrisPushGlobal

Variants: **IrisPushGlobalW**, **IrisPushGlobalH**

```
int IrisPushGlobal(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 IrisPushGlobal

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.

3.81 IrisPushGlobalH

Variants: **IrisPushGlobal**, **IrisPushGlobalW**

```
intIrisPushGlobalH(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 IrisPushGlobalH

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.82 IrisPushGlobalW

Variants: **IrisPushGlobal**, **IrisPushGlobalH**

```
int IrisPushGlobalW(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 IrisPushGlobalW

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.83 IrisPushGlobalX

Variants: **IrisPushGlobalXW**, **IrisPushGlobalXH**

```
int IrisPushGlobalX(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 IrisPushGlobalX

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.

3.84 IrisPushGlobalXH

Variants: **IrisPushGlobalX**, **IrisPushGlobalXW**

```
int IrisPushGlobalXH(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 IrisPushGlobalXH

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTAC	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.85 IrisPushGlobalXW

Variants: **IrisPushGlobalX**, **IrisPushGlobalXH**

```
int IrisPushGlobalXW(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 IrisPushGlobalXW

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTAC	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.86 IrisPushIEEEDbl

```
int IrisPushIEEEDbl(double num)
```

Arguments

num	Double value.

Description

Pushes an IEEE double onto the argument stack.

Return Values for IrisPushIEEEDbl

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_SUCCESS	The operation was successful.

3.87 IrisPushInt

```
int IrisPushInt(int num)
```

Arguments

num	Integer value.

Description

Pushes an integer onto the argument stack.

Return Values for IrisPushInt

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_SUCCESS	The operation was successful.

3.88 IrisPushInt64

```
int IrisPushInt64(long long num)
```

Arguments

num	long long value.

Description

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

Return Values for IrisPushInt64

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_SUCCESS	The operation was successful.

3.89 IrisPushList

```
int IrisPushList(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.

Return Values for IrisPushList

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a string element.

3.90 IrisPushLock

Variants: **IrisPushLockW**, **IrisPushLockH**

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

Arguments

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

Description

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

Return Values for IrisPushLock

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.91 IrisPushLockH

Variants: **IrisPushLock**, **IrisPushLockW**

```
int IrisPushLockH(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 **IrisAcquireLock** command by pushing the lock name on the argument stack.

Return Values for IrisPushLockH

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.92 IrisPushLockW

Variants: **IrisPushLock**, **IrisPushLockH**

```
int IrisPushLockW(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 **IrisAcquireLock** command by pushing the lock name on the argument stack.

Return Values for IrisPushLockW

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.93 IrisPushLockX

Variants: **IrisPushLockXW**, **IrisPushLockXH**

```
int IrisPushLockX(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 **IrisAcquireLock** command by pushing the lock name and an environment string on the argument stack.

Return Values for IrisPushLockX

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.94 IrisPushLockXH

Variants: **IrisPushLockX**, **IrisPushLockXW**

```
int IrisPushLockXH(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 **IrisAcquireLock** command by pushing the lock name and an environment string on the argument stack.

Return Values for IrisPushLockXH

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.95 IrisPushLockXW

Variants: **IrisPushLockX**, **IrisPushLockXH**

```
int IrisPushLockXW(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 **IrisAcquireLock** command by pushing the lock name and an environment string on the argument stack.

Return Values for IrisPushLockXW

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.96 IrisPushMethod

Variants: **IrisPushMethodW**, **IrisPushMethodH**

```
int IrisPushMethod(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 IrisPushMethod

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_BADARG	Invalid call argument.
IRIS_SUCCESS	The operation was successful.

3.97 IrisPushMethodH

Variants: **IrisPushMethod**, **IrisPushMethodW**

```
int IrisPushMethodH(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 IrisPushMethodH

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_BADARG	Invalid call argument.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.98 IrisPushMethodW

Variants: **IrisPushMethod**, **IrisPushMethodH**

```
int IrisPushMethodW(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 IrisPushMethodW

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_BADARG	Invalid call argument.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.99 IrisPushOref

```
int IrisPushOref(unsigned int oref)
```

Arguments

oref	Object reference.

Description

Pushes an OREF onto the argument stack.

Return Values for IrisPushOref

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERBADOREF	Invalid OREF.
IRIS_SUCCESS	The operation was successful.

3.100 IrisPushProperty

Variants: **IrisPushPropertyW**, **IrisPushPropertyH**

```
int IrisPushProperty(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 IrisPushProperty

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_BADARG	Invalid call argument.
IRIS_SUCCESS	The operation was successful.

3.101 IrisPushPropertyH

Variants: **IrisPushProperty**, **IrisPushPropertyW**

```
int IrisPushPropertyH(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 IrisPushPropertyH

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_BADARG	Invalid call argument.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.102 IrisPushPropertyW

Variants: **IrisPushProperty**, **IrisPushPropertyH**

```
int IrisPushPropertyW(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 IrisPushPropertyW

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_BADARG	Invalid call argument.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.103 IrisPushPtr

```
int IrisPushPtr(void * ptr)
```

Arguments

ptr	Generic pointer.

Description

Pushes a pointer onto the argument stack in internal format.

Return Values for IrisPushPtr

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.

3.104 IrisPushRtn

Variants: **IrisPushRtnW**, **IrisPushRtnH**

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

Arguments

rflag	Routine flags for use by **IrisDoRtn**
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 IrisPushRtnX 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 IrisPushRtn

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.

3.105 IrisPushRtnH

Variants: **IrisPushRtn**, **IrisPushRtnW**

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

Arguments

rflag	Routine flags for use by **IrisDoRtn**
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 IrisPushRtnXH 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 IrisPushRtnH

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.106 IrisPushRtnW

Variants: **IrisPushRtn**, **IrisPushRtnH**

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

Arguments

rflag	Routine flags for use by **IrisDoRtn**
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 IrisPushRtnXW 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 IrisPushRtnW

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.107 IrisPushRtnX

Variants: **IrisPushRtnXW**, **IrisPushRtnXH**

```
int IrisPushRtnX(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 **IrisDoRtn**
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 IrisPushRtn for a short form that only takes a tag name and a routine name.

Return Values for IrisPushRtnX

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.

3.108 IrisPushRtnXH

Variants: **IrisPushRtnX**, **IrisPushRtnXW**

```
int IrisPushRtnXH(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 **IrisDoRtn**
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 IrisPushRtnH for a short form that only takes a tag name and a routine name.

Return Values for IrisPushRtnXH

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.109 IrisPushRtnXW

Variants: **IrisPushRtnX**, **IrisPushRtnXH**

```
int IrisPushRtnXW(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 **IrisDoRtn**
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 IrisPushRtnW for a short form that only takes a tag name and a routine name.

Return Values for IrisPushRtnXW

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.110 IrisPushUndef

```
int IrisPushUndef()
```

Description

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

Return Values for IrisPushUndef

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_SUCCESS	The operation was successful.

3.111 IrisReleaseAllLocks

```
int IrisReleaseAllLocks( )
```

Description

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

Return Values for IrisReleaseAllLocks

IRIS_SUCCESS	The operation was successful.

3.112 IrisReleaseLock

```
int IrisReleaseLock(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 IRIS_IMMEDIATE_RELEASE and IRIS_SHARED_LOCK.

Description

Executes an InterSystems IRIS 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 IrisReleaseLock

IRIS_FAILURE	An unexpected error has occurred.
IRIS_SUCCESS	Successful lock.

3.113 IrisSecureStartA

Variants: **IrisSecureStartW**, **IrisSecureStartH**

```
int IrisSecureStartA(IRIS_ASTRP username, IRIS_ASTRP password, IRIS_ASTRP exename,
                      unsigned long flags, int tout, IRIS_ASTRP prinp, IRIS_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 InterSystems IRIS. 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 InterSystems IRIS. 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 InterSystems IRIS to set up a 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 connection, InterSystems IRIS does not open the principal input or output device until it is first used.

Valid values for the *flags* variable are:

- IRIS_PROGMODE — InterSystems IRIS 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 InterSystems IRIS results in closing the connection and returning error IRIS_CONBROKEN for both the current operation and any subsequent attempts to use Callin without establishing a new connection.)
- IRIS_TTALL — Default. InterSystems IRIS should initialize the terminal's settings and restore them across each call into, and return from, the interface.
- IRIS_TTCALLIN — InterSystems IRIS should initialize the terminal each time it is called but should restore it only when **IrisEnd** is called or the connection is broken.
- IRIS_TTSTART — InterSystems IRIS should initialize the terminal when the connection is formed and reset it when the connection is terminated.
- IRIS_TTNEVER — InterSystems IRIS should not alter the terminal's settings.
- IRIS_TTNONE — InterSystems IRIS 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.
- IRIS_TTNOUSE — This flag is allowed with IRIS_TTALL, IRIS_TTCALLIN, and IRIS_TTSTART. It is implicitly set by the flags IRIS_TTNEVER and IRIS_TTNONE. It indicates that InterSystems IRIS **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 IrisSecureStartA

IRIS_ACCESSDENIED	Authentication has failed. Check the audit log for the real authentication error.
IRIS_ALREADYCON	Connection already existed. Returned if you call **IrisSecureStartH** from a **$ZF** function.
IRIS_CHANGEPASSWORD	Password change required. This return value is only returned if you are using InterSystems authentication.
IRIS_CONBROKEN	Connection was formed and then broken, and **IrisEnd** has not been called to clean up.
IRIS_FAILURE	An unexpected error has occurred.
IRIS_STRTOOLONG	*prinp* or *prout* is too long.
IRIS_SUCCESS	Connection formed.

The flags parameter(s) convey information about how your C program will behave and how you want InterSystems IRIS to set terminal characteristics. The safest, but slowest, route is to have InterSystems IRIS 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 IRIS_TTNEVER requires the least overhead.

3.114 IrisSecureStartH

Variants: **IrisSecureStartA**, **IrisSecureStartW**

```
int IrisSecureStartH(IRISHSTRP username, IRISHSTRP password, IRISHSTRP exename,
                      unsigned long flags, int tout, IRISHSTRP prinp, IRISHSTRP 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 InterSystems IRIS. 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 InterSystems IRIS. 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 InterSystems IRIS to set up a 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 connection, InterSystems IRIS does not open the principal input or output device until it is first used.

Valid values for the *flags* variable are:

- IRIS_PROGMODE — InterSystems IRIS 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 InterSystems IRIS results in closing the connection and returning error IRIS_CONBROKEN for both the current operation and any subsequent attempts to use Callin without establishing a new connection.)
- IRIS_TTALL — Default. InterSystems IRIS should initialize the terminal's settings and restore them across each call into, and return from, the interface.
- IRIS_TTCALLIN — InterSystems IRIS should initialize the terminal each time it is called but should restore it only when **IrisEnd** is called or the connection is broken.
- IRIS_TTSTART — InterSystems IRIS should initialize the terminal when the connection is formed and reset it when the connection is terminated.
- IRIS_TTNEVER — InterSystems IRIS should not alter the terminal's settings.
- IRIS_TTNONE — InterSystems IRIS 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.
- IRIS_TTNOUSE — This flag is allowed with IRIS_TTALL, IRIS_TTCALLIN, and IRIS_TTSTART. It is implicitly set by the flags IRIS_TTNEVER and IRIS_TTNONE. It indicates that InterSystems IRIS **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 IrisSecureStartH

IRIS_ACCESSDENIED	Authentication has failed. Check the audit log for the real authentication error.
IRIS_ALREADYCON	Connection already existed. Returned if you call **IrisSecureStartH** from a **$ZF** function.
IRIS_CHANGEPASSWORD	Password change required. This return value is only returned if you are using InterSystems authentication.
IRIS_CONBROKEN	Connection was formed and then broken, and **IrisEnd** has not been called to clean up.
IRIS_FAILURE	An unexpected error has occurred.
IRIS_STRTOOLONG	*prinp* or *prout* is too long.
IRIS_SUCCESS	Connection formed.

The flags parameter(s) convey information about how your C program will behave and how you want InterSystems IRIS to set terminal characteristics. The safest, but slowest, route is to have InterSystems IRIS 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 IRIS_TTNEVER requires the least overhead.

3.115 IrisSecureStartW

Variants: **IrisSecureStartA**, **IrisSecureStartH**

```
int IrisSecureStartW(IRISWSTRP username, IRISWSTRP password, IRISWSTRP exename,
                      unsigned long flags, int tout, IRISWSTRP prinp, IRISWSTRP 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 InterSystems IRIS. 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 InterSystems IRIS. 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 InterSystems IRIS to set up a 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 connection, InterSystems IRIS does not open the principal input or output device until it is first used.

Valid values for the *flags* variable are:

- IRIS_PROGMODE — InterSystems IRIS 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 InterSystems IRIS results in closing the connection and returning error IRIS_CONBROKEN for both the current operation and any subsequent attempts to use Callin without establishing a new connection.)
- IRIS_TTALL — Default. InterSystems IRIS should initialize the terminal's settings and restore them across each call into, and return from, the interface.
- IRIS_TTCALLIN — InterSystems IRIS should initialize the terminal each time it is called but should restore it only when **IrisEnd** is called or the connection is broken.
- IRIS_TTSTART — InterSystems IRIS should initialize the terminal when the connection is formed and reset it when the connection is terminated.
- IRIS_TTNEVER — InterSystems IRIS should not alter the terminal's settings.
- IRIS_TTNONE — InterSystems IRIS 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.
- IRIS_TTNOUSE — This flag is allowed with IRIS_TTALL, IRIS_TTCALLIN, and IRIS_TTSTART. It is implicitly set by the flags IRIS_TTNEVER and IRIS_TTNONE. It indicates that InterSystems IRIS **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 IrisSecureStartW

IRIS_ACCESSDENIED	Authentication has failed. Check the audit log for the real authentication error.
IRIS_ALREADYCON	Connection already existed. Returned if you call **IrisSecureStartH** from a **$ZF** function.
IRIS_CHANGEPASSWORD	Password change required. This return value is only returned if you are using InterSystems authentication.
IRIS_CONBROKEN	Connection was formed and then broken, and **IrisEnd** has not been called to clean up.
IRIS_FAILURE	An unexpected error has occurred.
IRIS_STRTOOLONG	*prinp* or *prout* is too long.
IRIS_SUCCESS	Connection formed.

The flags parameter(s) convey information about how your C program will behave and how you want InterSystems IRIS to set terminal characteristics. The safest, but slowest, route is to have InterSystems IRIS 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 IRIS_TTNEVER requires the least overhead.

3.116 IrisSetDir

```
int IrisSetDir(char * dir)
```

Arguments

dir	Pointer to the directory name string.

Description

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

Return Values for IrisSetDir

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

3.117 IrisSetProperty

```
int IrisSetProperty( )
```

Description

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

Return Values for IrisSetProperty

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_SUCCESS	The operation was successful.

3.118 IrisSignal

```
int IrisSignal(int signal)
```

Arguments

signal	The operating system's signal value.

Description

Passes on signals caught by user's program to InterSystems IRIS.

This function is very similar to **IrisAbort**, but allows passing of any known signal value from a thread or user side of the connection to the InterSystems IRIS 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 InterSystems IRIS.

Example

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

Return Values for IrisSignal

IRIS_CONBROKEN	Connection has been broken.
IRIS_NOCON	No connection has been established.
IRIS_NOTINIRIS	The Callin partner is not in InterSystems IRIS at this time.
IRIS_SUCCESS	Connection formed.

3.119 IrisSPCReceive

```
int IrisSPCReceive(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 IRIS_ERFUNCTION will be returned.

Return Values for IrisSPCReceive

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERFUNCTION	Current device is not TCP device or is not connected.
IRIS_SUCCESS	The operation was successful.

3.120 IrisSPCSend

```
int IrisSPCSend(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 IRIS_ERFUNCTION will be returned.

Return Values for IrisSPCSend

IRIS_CONBROKEN	Connection has been closed due to a serious error.
IRIS_NOCON	No connection has been established.
IRIS_ERSYSTEM	Either the database engine generated a <SYSTEM> error, or Callin detected an internal data inconsistency.
IRIS_ERFUNCTION	Current device is not TCP device or is not connected.
IRIS_ERARGSTACK	Argument stack overflow.
IRIS_ERSTRINGSTACK	String stack overflow.
IRIS_SUCCESS	The operation was successful.
Any InterSystems IRIS error	From translating a name.

3.121 IrisStartA

Variants: **IrisStartW**, **IrisStartH**

```
int IrisStartA(unsigned long flags, int tout, IRIS_ASTRP prinp, IRIS_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 InterSystems IRIS. 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 InterSystems IRIS. 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 InterSystems IRIS to set up an InterSystems IRIS 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 connection, InterSystems IRIS does not open the principal input or output device until it is first used.

Valid values for the *flags* variable are:

- IRIS_PROGMODE — InterSystems IRIS 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

InterSystems IRIS results in closing the connection and returning error IRIS_CONBROKEN for both the current operation and any subsequent attempts to use Callin without establishing a new connection.)

- IRIS_TTALL — Default. InterSystems IRIS should initialize the terminal's settings and restore them across each call into, and return from, the interface.
- IRIS_TTCALLIN — InterSystems IRIS should initialize the terminal each time it is called but should restore it only when **IrisEnd** is called or the connection is broken.
- IRIS_TTSTART — InterSystems IRIS should initialize the terminal when the connection is formed and reset it when the connection is terminated.
- IRIS_TTNEVER — InterSystems IRIS should not alter the terminal's settings.
- IRIS_TTNONE — InterSystems IRIS 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.
- IRIS_TTNOUSE — This flag is allowed with IRIS_TTALL, IRIS_TTCALLIN, and IRIS_TTSTART. It is implicitly set by the flags IRIS_TTNEVER and IRIS_TTNONE. It indicates that InterSystems IRIS **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 IrisStartA

IRIS_ALREADYCON	Connection already existed. Returned if you call **IrisStartA** from a **$ZF** function.
IRIS_CONBROKEN	Connection was formed and then broken, and **IrisEndA** has not been called to clean up.
IRIS_FAILURE	An unexpected error has occurred.
IRIS_STRTOOLONG	*prinp* or *prout* is too long.
IRIS_SUCCESS	Connection formed.

The flags parameter(s) convey information about how your C program will behave and how you want InterSystems IRIS to set terminal characteristics. The safest, but slowest, route is to have InterSystems IRIS 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 IRIS_TTNEVER requires the least overhead.

Example

An InterSystems IRIS 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 an InterSystems IRIS backup. Output appears on the terminal.

```
IRIS_ASTR inpdev;
IRIS_ASTR outdev;
int rc;

strcpy(inpdev.str, "[BATCHDIR]dobackup");
inpdev.len = strlen(inpdev.str);
strcpy(outdev.str,"");
outdev.len = strlen(outdev.str);
rc = IrisStartA(IRIS_TTALL|IRIS_TTNOUSE,0,inpdev,outdev);
```

3.122 IrisStartH

Variants: **IrisStartA**, **IrisStartW**

```
int IrisStartH(unsigned long flags,int tout,IRISHSTRP prinp,IRISHSTRP 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 InterSystems IRIS. 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 InterSystems IRIS. 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 InterSystems IRIS to set up an InterSystems IRIS 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 connection, InterSystems IRIS does not open the principal input or output device until it is first used.

Valid values for the *flags* variable are:

- IRIS_PROGMODE — InterSystems IRIS 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 InterSystems IRIS results in closing the connection and returning error IRIS_CONBROKEN for both the current operation and any subsequent attempts to use Callin without establishing a new connection.)
- IRIS_TTALL — Default. InterSystems IRIS should initialize the terminal's settings and restore them across each call into, and return from, the interface.
- IRIS_TTCALLIN — InterSystems IRIS should initialize the terminal each time it is called but should restore it only when **IrisEnd** is called or the connection is broken.
- IRIS_TTSTART — InterSystems IRIS should initialize the terminal when the connection is formed and reset it when the connection is terminated.
- IRIS_TTNEVER — InterSystems IRIS should not alter the terminal's settings.
- IRIS_TTNONE — InterSystems IRIS 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.
- IRIS_TTNOUSE — This flag is allowed with IRIS_TTALL, IRIS_TTCALLIN, and IRIS_TTSTART. It is implicitly set by the flags IRIS_TTNEVER and IRIS_TTNONE. It indicates that InterSystems IRIS **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 IrisStartH

IRIS_ALREADYCON	Connection already existed. Returned if you call **IrisStartH** from a **$ZF** function.
IRIS_CONBROKEN	Connection was formed and then broken, and **IrisEndH** has not been called to clean up.
IRIS_FAILURE	An unexpected error has occurred.
IRIS_STRTOOLONG	*prinp* or *prout* is too long.
IRIS_SUCCESS	Connection formed.

The flags parameter(s) convey information about how your C program will behave and how you want InterSystems IRIS to set terminal characteristics. The safest, but slowest, route is to have InterSystems IRIS 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 IRIS_TTNEVER requires the least overhead.

Example

An InterSystems IRIS 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 an InterSystems IRIS 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 = IrisStartH(IRIS_TTALL|IRIS_TTNOUSE,0,inpdev,outdev);
```

3.123 IrisStartW

Variants: **IrisStartA**, **IrisStartH**

```
int IrisStartW(unsigned long flags,int tout,IRISWSTRP prinp,IRISWSTRP 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 InterSystems IRIS. 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 InterSystems IRIS. 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 InterSystems IRIS to set up an InterSystems IRIS 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 connection, InterSystems IRIS does not open the principal input or output device until it is first used.

Valid values for the *flags* variable are:

- IRIS_PROGMODE — InterSystems IRIS 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 InterSystems IRIS results in closing the connection and returning error IRIS_CONBROKEN for both the current operation and any subsequent attempts to use Callin without establishing a new connection.)
- IRIS_TTALL — Default. InterSystems IRIS should initialize the terminal's settings and restore them across each call into, and return from, the interface.
- IRIS_TTCALLIN — InterSystems IRIS should initialize the terminal each time it is called but should restore it only when **IrisEnd** is called or the connection is broken.
- IRIS_TTSTART — InterSystems IRIS should initialize the terminal when the connection is formed and reset it when the connection is terminated.
- IRIS_TTNEVER — InterSystems IRIS should not alter the terminal's settings.
- IRIS_TTNONE — InterSystems IRIS 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.
- IRIS_TTNOUSE — This flag is allowed with IRIS_TTALL, IRIS_TTCALLIN, and IRIS_TTSTART. It is implicitly set by the flags IRIS_TTNEVER and IRIS_TTNONE. It indicates that InterSystems IRIS **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 IrisStartW

IRIS_ALREADYCON	Connection already existed. Returned if you call **IrisStartW** from a **$ZF** function.
IRIS_CONBROKEN	Connection was formed and then broken, and **IrisEndW** has not been called to clean up.
IRIS_FAILURE	An unexpected error has occurred.
IRIS_STRTOOLONG	*prinp* or *prout* is too long.
IRIS_SUCCESS	Connection formed.

The flags parameter(s) convey information about how your C program will behave and how you want InterSystems IRIS to set terminal characteristics. The safest, but slowest, route is to have InterSystems IRIS 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 IRIS_TTNEVER requires the least overhead.

Example

An InterSystems IRIS 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 an InterSystems IRIS 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 = IrisStartW(IRIS_TTALL|IRIS_TTNOUSE,0,inpdev,outdev);
```

3.124 IrisTCommit

```
int IrisTCommit( )
```

Description

Executes an InterSystems IRIS TCommit command.

Return Values for IrisTCommit

IRIS_SUCCESS	TCommit was successful.

3.125 IrisTLevel

```
int IrisTLevel( )
```

Description

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

Return Values for IrisTLevel

IRIS_SUCCESS	TLevel was successful.

3.126 IrisTRollback

```
int IrisTRollback(int nlev)
```

Arguments

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

Description

Executes an InterSystems IRIS 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 IrisTRollback

IRIS_SUCCESS	TStart was successful.

3.127 IrisTStart

```
int IrisTStart( )
```

Description

Executes an InterSystems IRIS TStart command.

Return Values for IrisTStart

IRIS_SUCCESS	TStart was successful.

3.128 IrisType

```
int IrisType( )
```

Description

Returns the native type of the item returned by **IrisEvalA**, **IrisEvalW**, or **IrisEvalH** as the function value.

Return Values for IrisType

IRIS_ASTRING	8-bit string.
IRIS_CONBROKEN	Connection has been closed due to a serious error condition or **RESJOB**.
IRIS_DOUBLE	64-bit floating point.
IRIS_ERSYSTEM	Either ObjectScript generated a <SYSTEM> error, or if called from a **$ZF** function, an internal counter may be out of sync.
IRIS_IEEE_DBL	64-bit IEEE floating point.
IRIS_INT	32-bit integer.
IRIS_NOCON	No connection has been established.
IRIS_NORES	No result whose type can be returned (no call to **IrisEvalA** or **IrisEvalW** preceded this call).
IRIS_OREF	InterSystems IRIS object reference.
IRIS_WSTRING	Unicode string.

Example

```
rc = IrisType();
```

3.129 IrisUnPop

```
int IrisUnPop( )
```

Description

Restores the stack entry from **IrisPop**.

Return Values for IrisUnPop

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