Article
Robbie Luman · Feb 28, 2020 3m read

Using a Static C Library with the C Callout Gateway in Intersystems IRIS

Our company is in the process of converting our software for use in Intersystems IRIS and one of the major sections of the software makes use of a custom statically-linked C library using the $ZF("function-name") functionality. During this, I found out that the process for setting up the C library to be used within the database platform has changed significantly between Cache and IRIS.

If you have used or still use the C Callout Gateway with Cache, then you will know that in order for Cache to be made aware of any functions provided in a custom C library to be used with $ZF("function-name"), the library must be linked directly into the Cache kernel binary (see Cache 2012.1 documentation for this process: https://cedocs.intersystems.com/ens20121/csp/docbook/DocBook.UI.Page.cls?KEY=GCIOcallout#GCIOcallout_extapps). If any changes had to be made to the custom C library, this would mean having to re-link the changes into the Cache binary and bounce the Cache instance, which meant downtime for our applications.

Since this documentation was removed from the documentation from Cache 2012.2 on, it does not appear in the IRIS documentation at the moment (though I have been told that this new process may eventually be added back into IRIS documentation). So, here I will walk through how to implement a custom C function using $ZF("function-name"). This is a separate process than using the $ZF(-3,"function-name") which uses dynamically-linked libraries but, as it turns out, the $ZF("function-name") variety is pretty similar to it now compared to Cache.

The example below will depict a custom library called "simplecallout" to demonstrate how this process is carried out within IRIS. This example was performed in a Linux environment.

Note: Thank you to Intersystems Support for the assistance in identifying the correct procedure to enable this functionality in IRIS.

1. Create and save the source program "simplecallout.c". Example source looks like this:

```
//Include the IRIS callout header file
#define ZF_DLL
#include "iris-cdzf.h"

//Public function to add two integers
int AddTwoIntegers(int a, int b, int*outsum) {
  *outsum = a+b;  /* set value to be returned by the $ZF function call */
  return 0;  /* set the exit status code */
}

//IRIS directives to map the AddTwoIntegers function into a $ZF function
ZFBEGIN
  ZFENTRY("AddInt","iiP",AddTwoIntegers)
ZFEND
```

2. Compile the source to create the object (.o) file, including the include/ directory from the instance install directory (where the iris-cdzf.h file is located):

```
# gcc -c -fPIC simplecallout.c -I /intersystems/IRIR/dev/iris-
callin/include/ -o simplecallout.o
```

3. Generate a shared object (.so) file from the object file:

```
# gcc simplecallout.o -shared -o simplecallout.so
```

4. Copy and rename the shared object file into the <instanceinstalldir>/bin directory as "iriszf.so"

```
# cp simplecallout.so /intersystems/IRIS/bin/iriszf.so
```

5. From the IRIS programmer prompt, the function can now be used:

```
USER>write $zf("AddInt",1,4)
5
```

If changes need to be made to the library file, a new version of the shared object file is created and put in place of the existing iriszf.so file in the bin/ directory. These changes will be immediately available within IRIS without the need to restart the IRIS instance.

This new process also means that this linking process no longer has to be done each time a new instance is installed, and can simply be copied into the instance installation without the need for further action.

As this is my first post on the Developer Community, I hope that folks out there find this as useful and informative as I did in the discovery process of this new procedure.

#Callout #ObjectScript #Caché #InterSystems IRIS