How to make a LabVIEW CIN for Mac OS X using Xcode

Submitted by LabJack Support on Wed, 07/14/2010 - 16:07

**Update:** LabVIEW and Linux/Mac OS X documentation and downloads were updated on Sept. 7, 2010 in this blog post: [CiNs are out, Call Library Function Nodes are in!](http://labjack.com/articles/cins-out-call-library-function-nodes-are-in). We recommend reading this post instead since LabVIEW 2010 no longer supports CINs.

We have seen a few attempts from customers to try to create CINs for LabVIEW on Mac OS X. Here’s an example. We decided to try it ourselves, and it turned out be a little more complicated than we had originally expected. In order to help future customers, here are the steps we took to build a CIN:

For the sake of this tutorial, we have installed LabVIEW 2009 to the default location of:

```
/Applications/National Instruments/LabVIEW 2009/
```

and Xcode 3.2.2 installed in its default location:

```
/Developer
```

**Step 1: Get the CIN template to appear in Xcode**

Copy the “LabVIEW Templates” directory from

```
/Applications/National Instruments/LabVIEW 2009/cintools/Xcode Files
```

to:

```
/Developer/Platforms/MacOSX.platform/Developer/Library/Xcode/Project Templates
```

**Step 2: Create a blank CIN Project**

Launch Xcode and create a new project, there should now be a section called “LabVIEW Templates” with a template called “CIN 9.0”. Click that and give your project a name.
Step 3: Set the correct architecture

Right-click on your project’s blue icon and select “Get Info”. From there, click the “Build” tab.

In the section called “Architectures”, click the “Architectures” sub-section and select “Other”.

No description available.
Replace the value with “i386”.

Select “Mac OS X 10.6” from the “BaseSDK” sub-section.

Double click “Valid Architectures” and change it to “i386”.

Step 4: Add the Exodriver to the linker flags.

Scroll down in the “Build” tab until you find a section called “Linking”.

In the sub-section “Other Linker Flags”, enter the value “-llabjackusb”.

Step 5: Setup project search paths

Click the star in the bottom, left corner of the window, and select “Add User-Defined Setting”. 
Name the new setting `LABVIEW_CINTOOLS_PATH` and set its value to be the path (with quotes) to the cintools directory. In our case:

```
"/Applications/National Instruments/LabVIEW 2009/cintools/"
```

Repeat the same steps for `LABVIEW_MACHO_PATH`, with its value set to the path to the Mach-O directory. In our case:

```
"/Applications/National Instruments/LabVIEW 2009/cintools/Mach-O"
```

Scroll down in the “Build” tab until you find a section called “Search Paths”.

In the sub-section “Header Search Paths”, enter `$(LABVIEW_CINTOOLS_PATH)`. Leave the box labeled “recursive” unchecked.

In the sub-section “Library Search Paths”, enter `$(LABVIEW_MACHO_PATH)`. Leave the box labeled “recursive” unchecked.

Close the window.
Step 6: Select the correct build SDK

On the main project panel, just above the word “Overview” is a pull-down menu. Open it, and select “Mac OS X 10.6” from the “Active SDK” section.

Step 7: Update links to external files

Right-click on the file called “extcode.h” and select “Get Info”. If the path is red, click “Choose…” and point it to the extcode.h that lives in your LabVIEW’s cintools. In our case:

/Applications/National Instruments/LabVIEW 2009/cintools/extcode.h
Do the same thing for “liblabviewcin.a” and “liblvexports.a”, which can be found in the Mach-O directory. In our case:

/Applications/National Instruments/LabVIEW 2009/cintools/Mach-O/liblabviewcin.a
/Applications/National Instruments/LabVIEW 2009/cintools/Mach-O/liblvexports.a

**Step 8: Fix the target search paths**

In the section “Targets” from the main panel, right-click on the target you are going to build and select “Get Info”.

Click over to the “Build” tab. Scroll down to the “Search Paths” section.

In the “Header Search Paths” sub-section, replace the line with $LABVIEW_PATH with “$(LABVIEW_CINTOOLS_PATH)”.

In the “Library Search Paths” sub-section, replace the value with “$(LABVIEW_MACHO_PATH)”.

 DO NOT MODIFY THIS FILE. It is automatically generated.

    #include "platdefines.h"
    #include "fundtypes.h"
    #ifdef __cplusplus
    #define Cplusplus
    #endif

    #define this to keep C++ isms from littering the code #/
    #ifdef __cplusplus
    #endif

$(LABVIEW_CINTOOLS_PATH)
$(SYSTEM_DEVELOPER_DIR)/Headers/FlatCarbon
From there, just put your code inside ‘CINsource.c” and build. This should generate a .lsb file with the name of your project. This can then be loaded into a CIN from LabVIEW.

If you want to make a CIN that takes arguments, create a blankCIN and wire the inputs to it. Right-click on the CIN and select “Create .c File”. This will prompt you to save the file somewhere. Either import the .c file as the source for your project, or copy the contents of the .c file into “CINsource.c”.

**Step 9: Build the CIN**

Right-click on a target and select “Build”. This will build a .lsb that you can load from LabVIEW.

**Using LabVIEW CINs**

From LabVIEW, create a “Code Interface Node” and wire up the inputs. You can use the prototype in the .c file, if you are unsure of what the arguments are. Right-click on the CIN, select “Load Code Resource”, and select the .lsb. Now your LabVIEW VI can call that function.

**Download our Xcode project**

We used the steps above to build a suite of LabVIEWCINs to perform basic operations on a U3, such as analog input, digital I/O, and timers and counters. Download them here (Download removed). Our Xcode project is already configured, so downloading it will save you these steps.