LabChart for Windows

Automation

Revision 1.3

This page is intentionally left blank.

Contents

Specification Revision History

Revision	Who	Date	Description
1.0	JQ	12 April 2010	Creation
1.1	JQ	25 May 2010	Updated C++ section
1.2	IBK	28 June 2010	Typo removal
1.3	IBK	21 April 2011	LaTeX document

1 About this document

This document outlines methods with which to control LabChart from outside the LabChart application. It contains setup information and simple examples on scripting on various platforms. It is intended for LabChart users with some experience or interest in programming. It does not provide full support or debug information.

You will need:

- LabChart 7.1 or later
- Microsoft Visual Studio (if implementing in C++, CSharp or Visual Basic)
- Microsoft Excel (if implementing this)
- MATLAB (if implementing this)

2 Introduction: Automation using COM

Automation is an inter-process communication mechanism based on Component Object Model (COM) for the Windows platform. It provides the infrastructure for an application (the client) to control and access information from another application (the server) through shared automation objects exposed by the server. These automation objects are called type libraries.

The LabChart application can be controlled through external programs such as a simple command line tool written in C#, or customisable applications such as Microsoft Excel. Since the type library for LabChart is the same for any external application, you can use any of the functions used in the examples for all the different platforms.

If you would like to view the full type library interface to see what functions are available, you can import the LabChart type library into Microsoft Excel, and then view the interface in the Object Browser (F2) in the Visual Basic Editor. See the Excel section of this document for importing the type library and accessing the editor. If you would like more information on Automation or technical details not covered in this document, please see MSDN for official Microsoft documentation.

3 Visual Basic

Prerequisite: Before starting this example you need a Visual Basic development environment ready which can run a simple Visual Basic file. In Microsoft Visual Studio, select File → New →Project, and the select Console Application from the Visual Basic type templates under Other Languages.

Visual Basic is the simplest option for an external program to control LabChart. The first step is to gain access to the Automation interface by importing the LabChart type library. You can do this in Microsoft Visual Studio by right clicking on the project → Add Reference. Change to the COM tab, look for the ADInstruments LabChart 1.0 Type Library, and select OK.

In a suitable location in your project (such as the main subroutine in a new project), add the following lines to declare the application and document members.

```
 Dim chartApp As ADIChart.Application
 Dim chartDoc As ADIChart.Document
```

Now you can initialize these variables and use the LabChart interface to control behaviour. Add the following code to your subroutine. You will need to enter in a valid file path in order for the existing document example to work.

```
'Load existing document
 chartDoc = GetObject("C:\Program Files\ADInstruments\LabChart7\Welcome
    Center\Gallery\Getting Started\LabChart\Rabbit Respiration.adicht")

'Create new document
'chartDoc = GetObject("", "ADIChart.Document")
'chartDoc.Activate()

'Add a comment
chartDoc.AppendComment("Visual Basic Application Test Comment")

'Access the application name
 chartApp = chartDoc.Application
 MsgBox(chartApp.Name)
```

Additionally, you can make calls which are not part of the standard type library. These additional commands available can be viewed in the Macro recorder/editor in LabChart. If the action you desire can be recorded in the Macro editor, then you can paste them into your Visual Basic solution and alter the lines accordingly. For example, LabChart uses `Doc` as the member name in macros; you would need to change this to `chartDoc` for your code to work. The following line is an example.

```
chartDoc.ShowGraticule(False)
```

Add this line to your sub routine, OK the message box, and then see the graticule removed from Chart view. Manually, this is set in LabChart Setup → Display Settings.

4 Microsoft Excel

Prerequisite: In order to run macros from an Excel document, you need to save the document as a Macro-Enabled Workbook (available in the Save As dialog). In addition, you may need to allow macro content to be enabled in the document by responding to the application prompt.

In your document, press Alt+F11 to access the macro editor. Again you will need to import the LabChart type library in order to access the LabChart interface. Begin by selecting Tools → References, and then check ADInstruments LabChart 1.0 Type Library.

Now create a Module in the left hand Project pane by right clicking → Insert → Module. Modules are where you keep your utility subroutines so you can call them later on. Excel uses Visual Basic for its scripting so you could use the same code as in the Visual Basic example to achieve the same results. For now enter the following code into the module window.

```
Public Sub SampleAndSetSelection(doc As ADIChart.Document, samplingTimeSecs
    As Double, samples As Integer)
      Call doc.StartSampling(samplingTimeSecs, True)
      Call doc.AppendComment("End of sampling", 0)

      'insertion point at start of file (block, offset, block, offset)
      Call doc.SetSelectionRange(0, 0, 0, samples)
      'select channel 1
      Call doc.SelectChannel(0, True)
End Sub


Public Sub CopyDataPadCurrentRow(doc As ADIChart.Document, dest As Range,
    nCols As Long)
      'copy Data Pad values to Excel ranges
      Dim col As Long
      For col = 1 To nCols
            dest.Cells(1, col).Value = doc.GetDataPadCurrentValue(col)
      Next col
End Sub
```

There are two utility subroutines to use, one which samples for a given period and selects a set of data points and another which copies Data Pad values into a Range type.

Open up the editor for Sheet 1 (Sheet1) by double clicking on it, this will serve as your main subroutine. Add the following code.

```
Public Sub RunLabChart()
   Dim doc As ADIChart.Document

   'create new LabChart document
   Set doc = CreateObject("ADIChart.Document")

   'sample for 5 seconds in LabChart and select the first 1k samples (1 second
       of data at 1k/s)
   Call SampleAndSetSelection(doc, 5.0#, 1000)
   Call doc.AddToDataPad()

   'declare a range in the worksheet for the Data Pad values
   Dim dest As Range
   Set dest = Worksheets("Sheet1").Range("A1:D1")

   'copy the first 4 Data Pad columns to the range
   'in a new document, these will be the mean values for the first four
       channels in LabChart
```

```
        Call CopyDataPadCurrentRow(doc, dest, 4)
    End Sub
```

You can run the subroutine by pressing F5 or using the Run menu item. If you exit the Visual Basic editor and return to Excel, you will see that cells A1 to D1 have been filled with the values in the first Data Pad row.

Additionally, you can make calls which are not part of the standard type library. See the Visual Basic section of this document for more details.

5 MATLAB

Prerequisite: The MATLAB application.

In order to open a LabChart document you will need to use the `actxserver` function. Enter the following command in to MATLAB.

```
doc = actxserver('ADIChart.Document')
```

As in previous examples, this will give us access to the LabChart interface. For more examples on using MATLAB as an automation client for LabChart or Excel, view the MATLAB COM Client Support page in MATLAB documentation. For a simple example of LabChart automation in MATLAB, enter the following commands into a matlab file:

```
%sample 5 seconds of data in LabChart
doc.StartSampling(5, true)

%select 2000 sample and add that selection to the Data Pad
doc.SetSelectionRange(0, 0, 0, 2000)
doc.SelectChannel(0, true)
doc.AddToDataPad()

%retrieve the values from the first two columns in the data pad
meanVals = zeros(2);
for n=1:2
     x(n) = doc.GetDataPadCurrentValue(n)
end
```

6 C++

Prerequisite: Before starting this example you need a C++ development environment ready which can compile a simple C++ file. As an example, you can use an MFC template in Visual Studio by selecting File → New → Project, and the select MFC Application from the C++ project type templates. For the simplest example, use a dialog based application using the MFC Application Wizard, and copy code examples into the main entry function (`InitInstance`).

In order to expose the COM interfaces available in LabChart, the #import pre-processor directive must be used to import the LabChart type library. The type library contains all the information you need to call procedures in a DLL, in this case LabChart. Declare the following line at the top of the C++ file you are intending to work in (below the "stdafx.h" include if applicable).

```
#import "progid:ADIChart.Application" named_guids
using namespace ADIChart;
```

When your application is compiled using Visual Studio a LabChart7.tlh header file will now be automatically generated in your output directory. If you open this file now, you will see a list of functions has been generated that can be used to perform actions in LabChart. Several separate interfaces will have been created in this document, each providing a different set of functions. Adding the named_guids attribute to the end of the import directive is used to create named GUID constants (class and interface ids) which will be at the bottom of this header file. These ids will be used to create an instance of LabChart to control.

In this example, you will need to use a C++ macro to define a pointer type specifically for the LabChart interface. This macro has already been created in the LabChart7.tlh header file and will create the IADIChartDocumentPtr and IADIChartApplicationPtr types. You will also need to declare that you are using the ADIChart namespace (which is shown above with the using namespace directive) in order to access the pointer types and further constants.

In order for your application to use the LabChart interface, your program must first initialize the COM library. You can do this with the following line which needs to be added in an appropriate function or constructor.

```
//initialize the application for use with COM objects
HRESULT CI_HR = CoInitialize(NULL);
```

There are several ways you can create an instance of LabChart or a LabChart document. The first option is to use the `CoCreateInstance` function to create a new LabChart 7 process and initialize a pointer from which you can control further behaviour. Copy this code into your application in an appropriate location.

```
IADIChartDocumentPtr mChartDocI = 0;

//opens LabChart
HRESULT CCI_HR = CoCreateInstance(
CLSID_Document, //the class id, declared in LabChart7.tlh
 NULL,
CLSCTX_LOCAL_SERVER, //CLSCTX Enumeration, see MSDN for further information
 IID_IADIChartDocument, //the interface id, declared in LabChart7.tlh
(void**) &mChartDocI //the COM pointer to initialize
);

//create a new LabChart document
mChartDocI->Activate();
```

You must choose the correct CLSCTX flag for your application. The `CLSCTX\_LOCAL\_SERVER` used in the example is defined for an application in which the EXE code that creates and manages objects of this class runs on same machine but is loaded in a separate process space. This is suitable for this simple console application example. If this function fails, revise the MSDN page on CLSCTX flags and the HRESULT returned for more information.

When this code completes successfully, you will see that a new LabChart 7 instance has opened after `CoCreateInstance` has been called. If the `IADIChartApplication` interface id is used instead, LabChart will open with no new document.

You can use the following code in order to access the currently active instance of LabChart, and the active document.

```
IADIChartApplicationPtr mChartAppI = 0;
mChartAppI.GetActiveObject(CLSID_Application);
if (mChartAppI)
      mChartAppI->get_ActiveDocument(&mChartDocI);
```

Alternatively, you can use the CoGetObject function to open a pre-existing LabChart document or settings file.

```
//open an existing file
HRESULT hr = CoGetObject(_T("C:/Users/<username>/Documents/LabChart Gallery/
   My Data/<TestFile>.adicht"),
                    NULL,
                    IID_IADIChartDocument,
                    (void**) &mChartDocI
                    );

if (mChartDocI)
     mChartDocI->Activate();
```

Now that the `mChartDocI` variable has been initialized, you can use it to control LabChart through functions declared in the LabChart7.tlh header file. The following are some simple examples which you can use.

```
//start sampling in LabChart
//parameters: time to sample for
//             wait for sampling to end
//             sample mode stop constant (declared in header file)
mChartDocI->StartSampling(0, false, kSMUserStop);
//stop sampling in LabChart
mChartDocI->StopSampling();
//add a comment in LabChart to the first channel (indexed from 0)
//parameters: comment
//             channel index
mChartDocI->AppendComment(_T("Comment added by LabChart Automation Example")
   , 0);
```

Additionally, you can make calls which are not part of the standard type library. The commands available can be viewed in the Macro recorder/editor in LabChart. If the action you desire can be recorded, then you can use the `Invoke` command to issue that action in LabChart. This example will use an ATL class called `CComDispatchDriver` as a wrapper class for the IDispatch interface for which `Invoke` belongs to.

```
// Play a LabChart macro with one parameter
CComPtr<IDispatch> spDisp = CComQIPtr<IDispatch, &IID_IDispatch>(mChartDocI)
   ;
CComDispatchDriver spDrv(spDisp);
CComVariant varOnOff(L"false");

// Turn the LabChart Graticule off
HRESULT hr = spDrv.Invoke1(_T("ShowGraticule"), &varOnOff);
```

7 C#

Prerequisite: Before starting this example you need to have a C# development environment ready which can compile a simple C# file. In Microsoft Visual Studio, select File → New → Project, and the select Console Application from the Visual C# type templates under the Other Languages project type.

Automating LabChart in C# is a much simpler process to get started when compared to C++. Copy the following line into the top of your working document.

```
using System.Runtime.InteropServices;
```

Now you can access the LabChart type library. To do this in Microsoft Visual Studio, right click on References in the Solution Explorer and select Add Reference. Next, select the COM tab and look for the ADInstruments LabChart 1.0 Type Library. Select OK. In your main function you can now add the following code to access the LabChart application and document.

```
ADIChart.Document doc = null;
ADIChart.Application app = null;
```

You have the choice of opening up a new or existing document. Choose the option from the example below and paste into your program to access the document and application. You will need to enter in a valid file path in order for the existing document example to work.

```
 //open an existing file from the Welcome Center (Windows 7)
doc = (ADIChart.Document)Marshal.BindToMoniker("C:\\Program Files\\
    ADInstruments\\LabChart7\\Welcome Center\\Gallery\\Getting Started\\
    LabChart\\Rabbit Respiration.adicht");

////open a new file
//doc = new ADIChart.Document();
//doc.Activate();

//get running LabChart Application object
app = (ADIChart.Application)Marshal.GetActiveObject("ADIChart.Application");
```

Now that you have a document object to work with, you can perform actions in LabChart. The type library interface is the same as the interface used in the C++ section, but in the following example you can use LabChart to run a macro that detects and comments peaks in a file.

```
//insertion point at start of file.
doc.SetSelectionRange(0,0,0,0);
//select channel 1
doc.SelectChannel(0,true);
doc.PlayMacro("DetectPeaks");
int lastEndOffset = doc.SelectionEndOffset;
Console.WriteLine(lastEndOffset);
```

When you are finished using your COM variables you should release them with the following code.

```
if (app != null)
Marshal.ReleaseComObject(app);
if (doc != null)
Marshal.ReleaseComObject(doc);
```