

Introduction

Monday, February 8, 2016
11:17 AM

POP4 is a computer program that controls FMCW profiler radars developed by NOAA Earth System Research Laboratory, Physical Sciences Division.

The program sets the operating parameters of the radar, collects raw signal returns, processes these signals to produce derived data sets including returned power, Doppler shift, Doppler width, and auto- and cross-correlation functions (for space-antenna systems). An initial analysis of these data is also performed to produce various physical atmospheric data such as hydrometeorological melting layer heights and winds (under development).

The program can also read and reprocess radar data recorded by POPN, LapXM, and earlier POP programs -- any data file that is in the POP file format originally developed at NOAA Aeronomy Lab.

POP4 runs on a PC computer under Microsoft Windows versions from XP through Win8 (32- or 64-bit). The program itself is also compatible with Windows 10, but a current device driver incompatibility prevents it from working under Win10 on radars that use SpinCore PulseBlaster as a pulse generator device. Systems with a NOAA "pulsebox" card can only run POPN on 32-bit operating systems.

The program is written in C# and uses the Microsoft .NET environment. The program is compiled and built with Microsoft Visual Studio. The latest version used is Visual Studio 2013 Community Edition (free). The program is currently compiled to require that .NET Framework version 4.0 be installed on the computer in order to run POP4.

The program consists of two main parts: 1) a user control panel for setting radar and processing parameters and manually starting and stopping the radar and 2) the engine that actually controls the radar and processes the data. The control and processing engine normally runs as a separate Windows service. In that situation the control panel can be terminated, the user logged off, and the main program will continue to run. The service will automatically restart itself in the event of a power failure and computer restart. With a command line option, the control and processing engine can be made to run as a thread (not a separate process) under the control panel application. In this case killing the control panel will terminate the entire POPN program. This configuration is sometimes useful for debugging purposes.

Running POPN

Wednesday, February 17, 2016
6:52 PM

Instructions to run POPN4.

In the POPN executable directory run POPN4.exe.

The command line "POPN4" will start the POPN4 Control Panel and will run POPN4 as a separate Windows service.

The command line "POPN4 -noService" will start the POPN4 control Panel. The main POPN4 engine will run in the Control Panel program as a thread.

On the Control Panel main form, across the top are the control buttons for "Go", "Pause", and "Abort". Beneath these buttons is a status bar.

Beneath the status bar are the parameter selection controls.

Select a parameter file from the parameter selection dropdown box. If the files listed in the dropdown box are from the wrong folder, select another folder by clicking the button with the folder icon on it.

To view or modify the contents of the parameter file, click the "Parameters" button to bring up the [Parameter Setup Screen](#).

When ready to run with the specified parameter file, click the *Go* button. If the pause button is checked, the POPN program will pause at the end of the current dwell. Press *Go* again to continue. If the *Pause* button is not checked, the POPN program will run continuously. The *Abort* button will terminate the program in the middle of the current dwell. Pressing *Abort* when the program is *paused* will clear the current parameters and reset the program to its initial state.

Note that if the parameter file is modified while POPN is *paused*, the *Abort* button must be clicked before pressing *Go* again in order to unload the previous parameters and reinitialize the program. The Control Panel will pop up a reminder if you do not do this.

If POPN is running as a service, then once POPN is running (with *Pause* off) the Control Panel can be terminated. POPN will continue running (as a service). If an error occurs while running in this state, the service will try to stop and then restart itself. If this restart is successful POPN will continue running as before. Also, if the computer powers off and then back on, the POPN service will automatically restart and should continue running as before.

If POPN is not running as a service, then terminating the Control Panel program will terminate POPN.

Setting Parameters

Thursday, February 18, 2016
11:22 AM

Screen Shots

Wednesday, February 17, 2016
7:12 PM

POP 4.15 (2016/01/26)

GO ☐ Pause **Abort/Unload** ...

Stopped

..\parameters\sample.parx **Parameters**

Starting WITHOUT running as a service. (048 19:00:13)
Client: Connecting to comm server, thread 3 (048 19:00:13) (048 19:00:13)
Connected to comm server. (048 19:00:13)
Server: Comm Server connected to 1 client (048 19:00:14)
Server: Checking DDS device driver. (048 19:00:15)
Server: -No LibUsbK driver devices found. (048 19:00:15)
Server: Waiting for GO (048 19:00:15)

Service status: **Stopped** - Hide Options

Service

Start **Stop** **Uninstall**

Plots

Rx # 1 Ht Index 23 **Replot**

☒ Sampled Time Series ☐ Doppler Time Series at Ht
☐ Doppler A-Scan ☐ Clutter Wavelet Transform at Ht
☐ SNR Profile ☒ Doppler Spectrum at Ht
☐ Doppler Profile ☐ CrossCorrelation at Ht

Figure 1. The POPN Control Panel Main Form

PopNSetup: sample.parx

cuments\Visual Studio 2013\Projects\POP4\POP4\bin\parameters\sample.parx

Load Parx File Save

Undo All Changes Save As ...

Dwell Modes Output System Processing MeltingLayer **FM CW**

TX Sweep

IPP: 1200 usec

Center Freq: 60 MHz

Sweep Rate: 3644.011 Hz/usec

Freq Offset: 9765.662 Hz ☐

Gate Offset: 10.00 ☒

Delta Time Step: 11 Clk Per.

Number of Receivers

Using: 1 Attached: 0

Sample Parameters per IPP

Samples/FFT: 1024

NFFT: 1

Delay: 92400 nsec

Spacing: 1000 nsec

Window: Blackman

DC Filter: ☐ DC Filter2: ☐

Baseband Filter

☐ Apply Filter Gain Correction

☐ Filter Coeffs ☒ Gain Factors

Filter File: -Choose *.gain file-

DDS Control

☐ Enable

Ref Clock: 100 MHz

Multiplier: 4

System Clock: 400 MHz

Sync Clock: 100 MHz

Sync Clk Period: 10 nsec

--- Determined from DDS Registers ---

Sweep Rate: 3644.011 Hz/usec

Duration: 1126.400 usec

Offset: 9765.662 Hz

Tx DDS #1		Rx DDS #2	
57795737.59	Start F Hz	57805503.25	
620577007	(register)	620681865	
61900351.85	End F Hz	61910117.51	
664649967	(register)	664754825	

Delta Freq: 400.841 Hz

Delta T: 110.0 ns

Delta Freq: 4304 register

Delta T: 11 register

Derived Parameters

Sample Time: 9.8 sec

Range Res: 40.171 m

NHts: 11

First Gate: -0.402 km

Last Gate: 0.000 km

Gate Spacing: 267.99 ns

Last Sample: 1115 us

Dopp Nyquist: 22.03 m/s

Doppler Spectral Parameters

NPts: 4096

NSpec: 2

50% Overlap: ☐

Window: Blackman

DC Filter: ☒

Total Hts: 513

☒ Ht Indices to Keep: 0 thru 10

XCorr

1 XCorr NPts Multiplier

Max Lag: Poly Order: 200 7

Slope Fit Pts: Lags to Fit: 81 101

Fraction: LP Filter Lags Interp: 1.00 0

FFT Adjust Base: ☒ ☐

Start PBX Stop PBX

Figure 2. The Parameter Setup Form FM-CW Tab

PopNSetup: sample.parx

cuments\Visual Studio 2013\Projects\POP4\POP4\bin\parameters\sample.parx

Load Parx File Save

Undo All Changes Save As ...

Dwell Modes Output **System** Processing MeltingLayer FM CW

Station Name: DAC

Radar Name: FMCW

Radar Type

☐ Pulsed TX

☒ FM CW Dop

☐ FM CW SA

Edit System Parameters !

Latitude: 40.10 North

Longitude: 105.04 West

UT Correction: 0.00 hr (add to get UT)

Altitude: 1505 meters

Site ID: 508

TX Freq: 2835.00 MHz

Max Duty Cycle: 5000 %

Max TX: 15 usec

Min IPP: 200 usec

PBX Clock: 100 nsec

PreTR: 0 nsec

PostTR: 0 nsec

PreBlank: 0 nsec

PostBlank: 1100000 nsec

Sync: 500 nsec

Direction Definitions

Max Number of Directions: 9

Label	Azimuth	Zenith	Code
Vertical	0	90.00	0
East	90	76.45	1
North	0	76.45	2
-			
-			
-			
-			
-			
-			

Receivers

NRx: 1

Spacing (m): 0.946

Rx 1

Rx 2

Rx 3

A sub H: 2.635

Direction 2 to 1:

Refresh

Power Meter

☐ Enabled

Offset dB: 0.00

Interval sec: 0

Receiver BW Settings

	RX BW (ns)	Delay (ns)
0 :		0
1 :		
2 :		
3 :		

Figure 3. The Parameter Setup Form System Tab

PopNSetup: sample.parx

cuments\Visual Studio 2013\Projects\POP4\POP4\bin\parameters\sample.parx

Load Parx File Save

Undo All Changes Save As ...

Dwell Modes Output System Processing MeltingLayer FM CW

Replay Mode

☐ Reprocess Recorded Data DY400621000h.SPC

Reprocess beginning with

☐ Raw Samples ☐ Doppler Time Series

☐ Doppler Spectra ☐ CrossCorrelation

☐ Moments ☒ No Recalc

☐ Use FMCW NSpec for TIME SERIES processing

1 Number of records to average spectra over

☐ Do Autocorr on Single Rx (Setup on FMCW page)

Select File to Play Back:

C:\Data\Wallops\spectra\DY4006210 Browse

Start time: 00:00 Start day:

End time: 24:00 End day:

Clutter Removal

☐ Exclude clutter up to: 3.000 km AGL

Output files have ☒ Original spectra ☐ Clutter deleted from spectra

Parameters

		Default
PercentLess	0	20
TimesBigger	0.0	3
Restrict GC Extent	<input type="checkbox"/>	On
Restrict If Prev DC	<input type="checkbox"/>	On
Min SNR at DC (dB above ACR)	0.0	-99 dB (off)
Replace with:	0	(0=noise, 1=interp)

Debug Options

☒ No DAQ (create fake data samples)

☒ No PBX (trigger DAQ internally)

☒ Use Memory Allocator..... 5 Block size

☐ Write Debug Log File

☐ Alloc Time Series Only

☐ Save Filtered Time Series

☐ Do Parallel Tasks

Restrict Spectral Moment Intervals

Search for signal peak only between:

☐ Enable Low Doppler: -1.00 m/s

High Doppler: 7.00 m/s

Consensus Averaging

☐ Enable Mode

Avg Time: minutes

Oblique Vertical

Threshold: % %

Window: m/s m/s

☒ Vertical Correction

Cns File Folder:

Browse

Wavelets

	Thld/median	Cutoff m/s	# Gates
<input type="checkbox"/> Clutter (D20)	1.0	0.5	20.0
<input type="checkbox"/> DeSpike (D2)	10.0		
<input type="checkbox"/> Birds (Harmonic)			

Figure 4. The Parameter Setup Form Processing Tab

PopNSetup: sample.parx

cuments\Visual Studio 2013\Projects\POP4\POP4\bin\parameters\sample.parx

Load Parx File Save

Undo All Changes Save As ...

Dwell Modes Output System Processing MeltingLayer FM CW

Output File Options

☒ **Write Data to Disk File A**

Data to Write

☒ Spectra ☐ Single Time Series ☐ Text ☒ Daily Files ☒ Append

☒ Moments ☐ Full Time Series ... ☐ Text ☐ Hourly Files ☐ Overwrite

☐ CrossCorr ☐ Raw TS File ... ☐ Text

File Naming Convention

☐ POP (Dyydddx.SPC) ☒ LAPXM (Dssyydddx.SPC)

Suffix (x): Site (sss):

Folder: Browse for Folders

File:

Log Folder: Browse for Folders

Output File Options

☐ **Write Data to Disk File B**

Data to Write

☐ Spectra ☐ Single Time Series ☐ Text ☐ Daily Files ☒ Append

☐ Moments ☒ Full Time Series ... ☐ Text ☒ Hourly Files ☐ Overwrite

☐ CrossCorr ☐ Raw TS File ... ☐ Text

File Naming Convention

☐ POP (Dyydddx.SPC) ☒ LAPXM (Dssyydddx.SPC)

Suffix (x): Site (sss):

Folder: Browse for Folders

File:

Figure 5. The Parameter Setup Form Output Tab

PopNSetup: sample.parx

cuments\Visual Studio 2013\Projects\POP4\POP4\bin\parameters\sample.parx

Load Parx File Save

Undo All Changes Save As ...

Dwell Modes Output System Processing MeltingLayer FM CW

Melting Layer Algorithm

☐ Enable

Time Interval Minutes: 10

MinHeightM: 600

MaxHeightM: 6000

MinSnrDvvPairs: 4

MinSnrRain: 40 *

DeltaSnrBb: 2.5

DeltaDvvBb: -1.5

MinSnrBb: 50 *

DvvBbOnlyMaxHeightM: 300

DvvBbOnlyMinSnr: 42 *

GateSpacingResolution: 5

BrightBandPercent: 25

AcceptHeightRangeM: 211

AcceptPercent: 65

QcMaxRainAboveBb: 10

Output File Folder: D:\Etldata\RadarmcW\Hourly Browse for Folders

☒ Hourly Files? ☐ Include Header

Write Log File to Folder: ☒ D:\Etldata\RadarmcW\SpecMom Browse for Folders

☒ Use data only in this region:

MinDataHtM: 140

MaxDataHtM: 6000

* ☒ Use range-adjusted SNR threshold

Offset value: -15.0 dB

☐ Modify all SNRs with upper gate noise level

Noise gate index range: 0 to 0

☒ Skip gates with narrow widths

Min width: 0.30 m/s

Figure 6. The Parameter Setup Form Melting Layer Tab

PopNSetup: sample.parx

cuments\Visual Studio 2013\Projects\POPN4\POPN4\bin\parameters\sample.parx

Load Parx File Save

Undo All Changes Save As ...

Dwell Modes Output System Processing MeltingLayer FM CW

DAC FMCW

FM CW Doppler Radar 2835.000 MHz

Beam Parameters

☒ TX Pulse (ON) # Sets: 1

	#1
IPP (usec):	1200
PW (nsec):	268
Code Bits:	0
Phase Flip:	0
Nhts:	11
1st Sample (nsec):	-2680
Spacing (nsec):	268
Coh. Avg:	1
Spectral Avg:	2
Npts:	4096
Full Scale (m/s):	
Dwell (sec):	
First Range (km):	
Last Range:	
IPP (km):	
Pulse Res (m):	

Antenna Beam Sequence

Sets: 10

NReps:	1	1	1							
Vertical	1									
East		1								
North			1							
—										
—										
—										
—										
—										
—										

The columns above show the parameter set, the direction, and the number of repetitions for each sequential beam position in the dwell cycle.

Figure 7. The Parameter Setup Form Dwell Modes Tab. Not used for FM-CW systems -- only pulsed Doppler radars.

State File

Wednesday, February 24, 2016
12:56 AM

Compiling and Building with Visual Studio

Wednesday, February 24, 2016

4:55 PM

VS: Table of Contents

Friday, February 19, 2016
4:38 PM

[Introduction](#)

[POPN4 Solution](#)

[POPN4ControlPanel Project](#)

[POPN4Service Project](#)

[PopCommunication Project](#)

[Intel IPP Projects](#)

[Intel Libraries](#)

[NOAA Libraries](#)

[DACarter.Utilities.dll](#)

[DACarter.ClientServer.dll](#)

[DACarter.PopUtilities.dll](#)

[DACarter.NOAA.dll](#)

[DACarter.Utilities.Maths.dll](#)

[DACarter.Utilities.Graphics.dll](#)

[DACarter.NOAA.Hardware Libraries](#)

[DAQDevice.dll](#)

[PulseGenDevice.dll](#)

[AD9959EvalBdUsbK.dll](#)

[MCPowermeter.dll](#)

[ZGraphDac.dll](#)

[Other Libraries and Programs](#)

[Numerical Methods Library](#)

[MathNet Library](#)

[ZedGraph Graphics Library](#)

[Progressbar.dll](#)

[Fx2loader.exe](#)

[TVicPort.dll](#)

[SpinAPI.NET.dll](#)

Introduction

Friday, February 12, 2016
4:05 PM

This section describes the files and procedures to compile and build the POPN4 program.

Microsoft Visual Studio 2013 Community Edition is the current version used, but it should be straightforward to move all projects over to Visual Studio 2015.

The POPN4 program consists of

1. A [Visual Studio solution](#) (POPN4.sln) and its included projects.
2. Various NOAA/PSD [class library projects](#).
3. Intel [IPP libraries](#).
4. Other [third party libraries](#) and programs.

Load the solution that you want to build into Visual Studio.

Click on "Build Solution" (F6) to build all the projects within the solution.

Note: When building POPN4 solution the POPNService must be stopped in order to update the code for the service.

For those who are trying to follow the source code and who are not that familiar with Visual Studio, it can be very useful to right click on a method or other item and select "Go To Definition" to find its definition or select "Find all References" to see every place that item was used in the program.

Note on project structure within Visual Studio:

When you add a source code file to the project, the project file will have entries such as:

```
<Compile Include="MyClass1.cs" />
```

```
<Compile Include="MyClass2.cs" />
```

and the solution explorer will show each file on a separate line.

But some files can be made dependent on others, so that they show up as subitems that can be hidden.

Visual Studio does this for Windows Forms files, for example, where VS created files

MyForm.Designer.cs and

MyForm.resx

will show up as subitems beneath the user code file MyForm.cs .

I manually did the same thing when I split the code in PopNDwellWorker.cs into subfiles.

To do this, in the project file I changed

```
<Compile Include="PopNDwellWorker.cs" />
```

```
<Compile Include="PopNDwellWorker.Plotting.cs" />
```

to

```
<Compile Include="PopNDwellWorker.cs" />
```

```
<Compile Include="PopNDwellWorker.Plotting.cs">
```

```
<DependentUpon>PopNDwellWorker.cs</DependentUpon>
```

```
</Compile>
```

POP4 Solution

Monday, February 8, 2016
12:42 PM

The POPN program itself is built as a Visual Studio solution, `POP4.sln`, typically kept in the folder
...\\Documents\\Visual Studio\\projects\\POP4.

The POP4 solution consists of 7 projects found in 7 subfolders of the main POP4 directory:

```
.\POP4\POP4ControlPanel.csproj ,  
.\POP4Service\POP4Service.csproj ,  
.\POPCommServer\POPCommunication.csproj ,  
.\IppDlls\ippch_cs\ippch_cs.csproj ,  
.\IppDlls\ippcore\ippcore.csproj ,  
.\IppDlls\Ippdefs\ippdefs.csproj , and  
.\IppDlls\ipps\ipps.csproj .
```

The 4 *IppDlls* projects are for the Intel Integrated Processing Primitives (IPP) library functions that optimize computations for the CPU being used. Their source code is provided by Intel and so these libraries do not generally change or need rebuilding. These libraries may be referenced by the 3 main POPN projects.

POP4ControlPanel is set as the startup project for the POP4 solution (under solution properties). That means when you "run" the solution, the POP4ControlPanel starts up.

Property settings for all projects:

Target Framework: .NET Framework 4

Configuration: Debug

Platform: Any CPU

Platform Target: Any CPU

Debug Start Options, Command Line argument: `-noService`

POP4ControlPanel output is a Windows application.

POP4Service is a console application.

All other projects are class libraries.

To change the Windows file version number of each project output file, you must manually edit the assembly version number either in *AssemblyInfo.cs* (under Properties in Solution Explorer) or in *Application > Assembly Information* in the project properties. For POPN version 4.15 set the assembly version to `4.15.*.*`, for example.

After compiling and building the solution, the executable files are found in
...\\projects\\POP4\\POP4\\bin\\debug.

POPN4ControlPanel Project

Thursday, February 11, 2016
12:47 PM

Project folder: ...\\projects\\POPN4\\POPN4

Properties:

Application > Assembly Name: POPN4
Output type: Windows Application
Icon: POPN4.ico
Manifest: Embed manifest with default settings
Build > Platform Target: Any CPU
Debug > Start Options > Command line arguments: -noService
(This option means when debugging from within Visual Studio, the code in POPN4Service will run as a thread within the POPN4ControlPanel application, not as a service.)

References:

ippdefs_cs.dll
PopCommunication.dll
DACarter.ClientServer.dll
DACarter.Utilities.dll
DACarter.Utilities.Graphics.dll
DACarter.PopUtilities.dll
AD9959EvalBd.dll
DAQDevice.dll
PulseGenDevice.dll
Framework.Controls.ProgressBar.dll
PopN4Service.exe

Project Files:

POPN4.ico (Build Action = embedded resource)
Program.cs
POPN4MainForm.cs
PopNSetup3.cs
Settings.cs
SaveChangesBox.cs
SequenceForm.cs
PowerMeterDisplay.cs

In Solution Explorer, the References section for this project includes not only all the class libraries referenced, but also a reference to POPN4Service.exe (with Local Copy = true), in order to make sure the latest compiled version of POPN4Service is in the executable directory for POPN4.exe.

The POPN4ControlPanel project contains several "application settings" that are defined in Visual Studio under project Properties -> Settings. The programmer defines these in Visual Studio, outside of the source code. All settings have a name, type, and initial or default setting. All are user scope. When the project is built, these settings are coded into the source file *Settings.Designer.cs*, which is created and modified by Visual Studio (user should not edit this file). These application settings are used to persist or remember between successive invocations of the POPN program certain user-selected options on the main user control panel. Also, the location and size of various display windows are remembered from the previous run of the program.

The initial settings are stored in the file *app.config* at design time and the user selected settings are stored in the file *user.config* at run time.

These settings actually get buried in a location similar to this:

C:\\Users\\Dave\\AppData\\Local\\POPN4\\POPN4.exe_Url_4newwi1fpn2dvvhndbdvwnfzdr5v4td\\4.15.5885.26509\\user.config

But there is no need to know about this, it is all handled internally by Visual Studio and POPN when it runs.

POPN4Service Project

Thursday, February 11, 2016
3:30 PM

Project Folder: ...\\POPN4\\POPN4Service

Properties:

Application > Assembly Name: POPN4Service
Output type: Console Application
Build > Platform Target: Any CPU

References:

lppdefs_cs.dll
lpps_cs.dll
PopCommunication.dll
DACarter.ClientServer.dll
DACarter.Utilities.dll
DACarter.Utilities.Maths.dll
DACarter.PopUtilities.dll
DACarter.NOAA.dll
AD9959EvalBdUsbK.dll
DAQDevice.dll
MathNet.Iridium.dll
MCPowermeter.dll
PulseGenDevice.dll

Project Files:

POPN4Service.cs
ServiceStarter.cs
PopNDwellworker.cs
 PopNDwellworker.DopplerTS.cs
 PopNDwellworker.Spec.cs
 PopNDwellworker.Plotting.cs
PopDataPackage3.cs
PopNReplay3.cs
PopFileWriter.cs
PopNAllocator.cs
PopNConfig.cs
PopNConsensus.cs
GroundClutter3.cs
IntelIPP3.cs (obsolete)
LoadDDSFirmware.cs
MeltingLayerCalculator.cs
PopSequencer.cs

PopCommunication Project

Thursday, February 11, 2016
3:38 PM

Project folder: ...\\projects\\POPN4\\POPCommServer

Properties:

Application > Assembly Name: POPCommServer
Output type: Class Library
Build > Platform Target: Any CPU

References:

DACarter.PopUtilities.dll
DACarter.ClientServer.dll
ippdefs_cs.dll

Project Files:

PopCommunication.cs

This library defines the following classes to handle the WCF communication between the Control Panel and the POPN Service:

PopCommunicator
PopCommClient
PopCommServer
PopCommServerHost

See [Communication](#) section of this document.

Intel IPP Projects

Thursday, February 11, 2016
3:44 PM

There are 4 separate projects that build Intel Integrated Processing Primitives (IPP) libraries:

```
ippdefs_cs.dll  
ippch_cs.dll  
ippcore_cs.dll  
ipps_cs.dll
```

The first defines various IPP data types. The others direct various C# function calls to the proper CPU-specific Intel libraries.

Intel Libraries

Thursday, February 11, 2016
3:43 PM

32-bit libraries:

ippcore-6.1.dll
ipps-6.1.dll
ippsp8-6.1.dll
ippsp8-6.1.dll
ippst7-6.1.dll
ippsv8-6.1.dll
ippsw7-6.1.dll
libiomp5md.dll

64-bit Libraries:

ippcoreem64t-6.1.dll
ipps-6.1.dll
ippse9-6.1.dll
ippsm7-6.1.dll
ippsmx-6.1.dll
ippsn8-6.1.dll
ippsu8-6.1.dll
ippsy8-6.1.dll
libiomp5md.dll

Before running POPN, the proper Intel IPP libraries must be copied to the executable directory. These files are kept in 2 distinct folders on the installation media. Note that 2 of the files have the same name in 32-bit and 64-bit versions (ipps-6.1.dll and libiomp5md.dll). In the storage folders, there are copies of these files with names appended with 32 or 64. If the properly named files get overwritten with the wrong version, these backup files can be renamed to the proper name.

These Intel libraries should never need to be recreated. However, the Intel IPP installation programs are included in the POPN Extras folder on the installation media. Follow these instructions to create these libraries.

Install IPP version 6.1.5 via files in installation folder:

w_ipp_em64t_p_6.1.5.054.exe
w_ipp_ia32_p_6.1.5.054.exe

This creates folder in Program Files\Intel

Look in ipp-samples\language-interface\csharp\interface\src

Create C# projects for ippdefs_cs.dll and ipps_cs.dll from source files

ippdefs.cs and ipps.cd (**NOTE:** these projects are now part of POPN4 solution.)

Then reference these dlls in projects that use Intel IPP

ippdefs_cs.dll
ipps_cs.dll

Then at run-time need Pre-built Intel IPP dlls in executable path.

ipps-6.1.dll (or ippsem64t-6.1.dll renamed)
ippcore-6.1.dll
libiomp5md.dll

plus

ipps??-6.1.dll

where ?? represents CPU-specific versions

such as p8, px, s8, t7, v8, w7 for ia32
and e9, m7, mx, n8, u8, y8 for em64t

These are found in Program Files\Intel\IPP bin folder for version and CPU

The dispatcher dll (ipps-6.1.dll) will call the dll that best matches the CPU at runtime.

The 64-bit install does not create ipps-6.1.dll -- you must rename ippsem64t-6.1.dll.

NOAA Libraries

Thursday, February 11, 2016
3:40 PM

[DACarter.ClientServer.dll](#)

[DACarter.PopUtilities.dll](#)

[DACarter.NOAA.dll](#)

[DACarter.Utilities.dll](#)

[DACarter.Utilities.Maths.dll](#)

[DACarter.Utilities.Graphics.dll](#)

[DACarter.NOAA.Hardware libraries:](#)

~~[DACarter.NOAA.Hardware.dll](#)~~

[DAQDevice.dll](#)

[PulseGenDevice.dll](#)

[AD9959EvalBdUsbK.dll](#)

[MCPowermeter.dll](#)

DACarter.Utilities.dll

Friday, February 12, 2016
2:45 PM

References:

No other NOAA libraries are referenced.

Project Files:

Contains many generally useful utility class files.

DACarter.ClientServer.dll

Friday, February 12, 2016
2:44 PM

References:

DACarter.Utilities.dll

Project Files:

DacClientServer.cs

Contains basic classes for developing WCF client/server with duplex messaging.

ServiceControllerHelper.cs

Classes for controlling a Windows service.

ServiceHelper.cs

Classes to help set up a custom Windows service.

DACarter.PopUtilities.dll

Friday, February 12, 2016
2:43 PM

References:

DACarter.Utilities.dll
ippdefs_cs.dll

Project files:

PopParameters.cs

Class that defines all POPN parameters.

PopDataPackage3.cs

Class that wraps all the data produced by a dwell along with the entire parameter set and timestamp.

PopConfig.cs

Handles the PopStateFile, which contains all the configuration data to enable restarts after power failures, etc.

DacLogger.cs

Class the makes it easy to write a time-stamped entry in a generic log file.

DACarter.NOAA.dll

Friday, February 12, 2016
2:45 PM

References:

DACarter.Utilities.dll

Project Files:

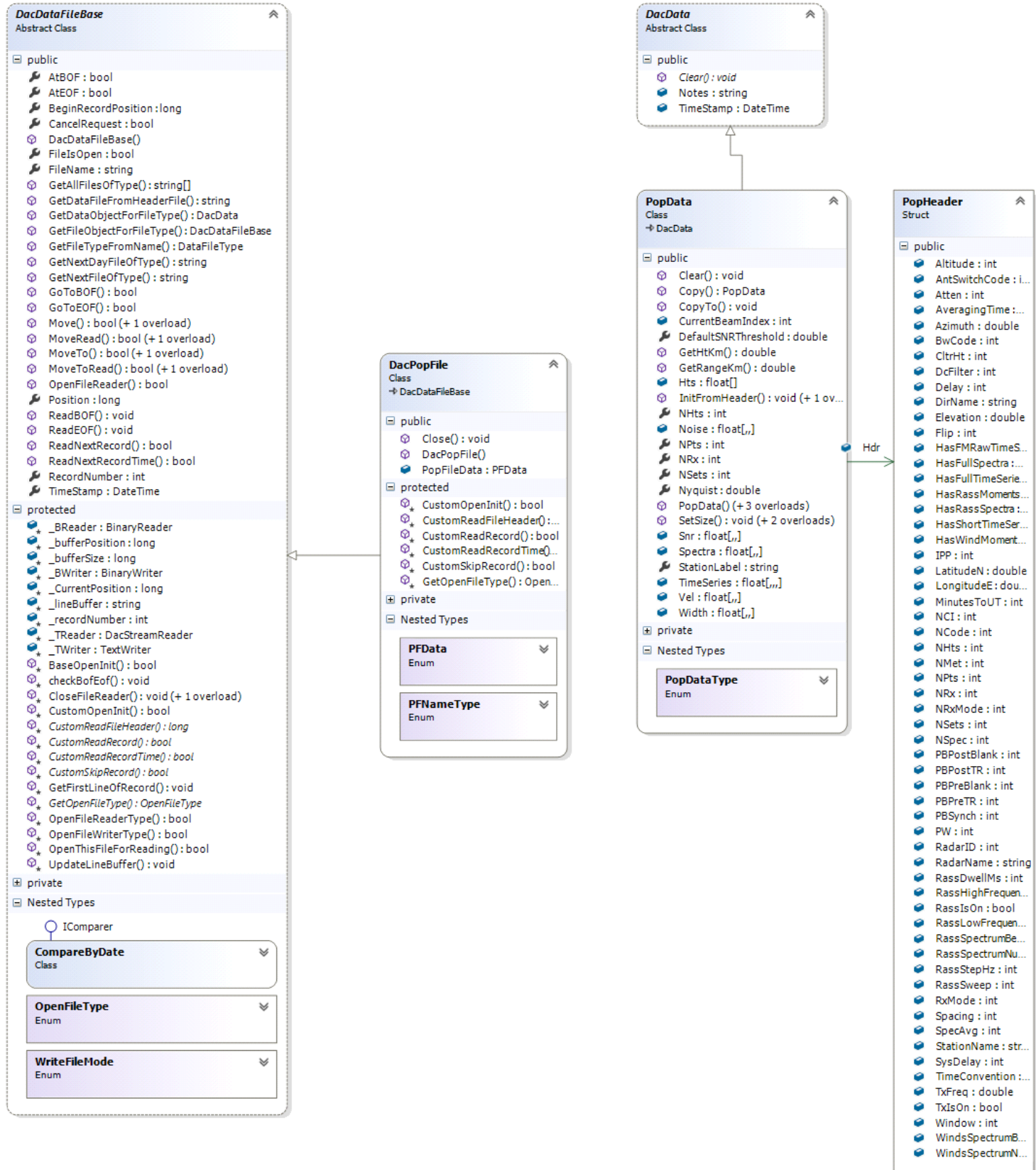
Many classes to handle data specific to NOAA profilers and other instruments.

In particular, POPN uses these classes to read POP format files and to handle the data blocks from these files:

DacPopFile (derived from *DacDataFileBase*) and
PopData (derived from *DacData*)

Class Diagrams

Thursday, February 18, 2016
1:51 PM



DACarter.Utilities.Maths.dll

Friday, February 12, 2016
2:45 PM

References:

- DACarter.Utilities.dll
- DACarter.Utilities.Graphics.dll
- MathNet.Iridium.dll
- MathNet.Numerics.dll
- NumericalMethodsLibrary.dll
- ippdefs_cs.dll
- ipps_cs.dll

Project Files:

CurveFit.cs

Classes for fitting curves to Gaussians and polynomials.

FFT.cs

A static FFT class that wraps FFT routines found in the *MathNet* library. These routines operate on arrays of type *double* and type *MathNet.Numerics.Complex*.

HSMMethod.cs

Contains a static *HSMMethod.Noise* method that calculates spectral noise level using the Hildebrand and Sekhon method. Note that this method has been superseded by the *Moments.GetNoise* method (which also uses the HS technique).

IntelMath.cs

Various math functions that use the Intel IPP math library -- in particular, FFT and linear least squares algorithms. These methods operate on types *double* and *ipp64fc* (Intel's complex type). POPN uses these IntelMath FFT routines.

Moments.cs

Contains static methods to calculate moments of spectra (including noise level). These methods use the built-in .NET System.Math library.

Wavelet.cs

Contains classes to compute various wavelets: harmonic, continuous, Daubechies. Note that POPN uses the DaubechiesWavelet class, not the DaubechiesWaveletDac class.

DACarter.Utilities.Graphics.dll

Friday, February 12, 2016
2:45 PM

References:

ZGraphDac.dll

Project Files:

QuickPlotZ.cs

QuickPlotForm.cs

ColorPropertyForm.cs

The QuickPlotZ class uses the ZedGraph graphics library (wrapped in ZGraphDac library) to make simple plots.

Class Diagram

Thursday, February 18, 2016
3:43 PM

QuickPlotZ
 Class

Fields

Properties

- CommentLabel
- Form
- GraphControl
- PaneFill
- PlotColorScale
- TypeOfPlot

Methods

- _form_FormClosed
- AddCurve (+ 3 overloads)
- addX2Axis
- AddX2Axis
- ClearLegend
- ClearPaneCurves
- ClearPlot
- ColorBoxPlot
- ColorContourPlot
- ColorMenuItemHandler
- ColorSurfacePlot
- ContourSurfacePlot
- Display
- DrawColorSurfacePlot
- DrawContourGraph
- DrawContourSurfacePlot
- DrawFilledBoxGraph
- DrawLegend
- GetHashCode
- getPosition
- Hide
- InitGraphObjects
- MyContextMenuBuilder
- MyPointValueHandler
- QuickPlotZ
- setPosition
- setSize
- SetTitles (+ 1 overload)
- SetWindowTitle
- StackedPlot (+ 1 overload)

Nested Types

ContourBox
 Class

PlotType
 Enum

QuickPlotForm
 Class
 → Form

Fields

- FitGraphToWindow
- FixedAspectRatio
- IsResizing

Properties

- GraphControl

Methods

- Dispose
- Form1_ClientSizeChanged
- Form1_FormClosing
- Form1_Load
- InitializeComponent
- QuickPlotForm
- QuickPlotForm_FormClosed
- QuickPlotForm_Paint
- QuickPlotForm_ResizeBegin
- QuickPlotForm_ResizeEnd

DACarter.NOAA.Hardware Libraries

Friday, February 12, 2016
2:45 PM

DAQDevice.dll

Friday, February 12, 2016
2:48 PM

References:

- DAcarter.Utilities.dll
- DAcarter.PopUtilities.dll
- MccDaq.dll
- DAQCommLib.dll

Project Files:

- DAQDevice.cs
 - Abstract base class for MCC and IOTech boards.
- DAQBoardIOTech.cs
 - Class for IOTech board.
- DAQBoardMCC.cs
 - Class for MCC board.
- DAQBoardMCCFile.cs
 - Class that creates the file
C:\Program Files\Measurement Computing\DAQ\CB.CFG
which is used by the MCC drivers.

See DAQ [installation notes](#).

PulseGenDevice.dll

Friday, February 12, 2016
2:48 PM

References:

DACarter.Utilities.dll ,
DACarter.PopUtilities.dll ,
[SpinAPI.NET.dll](#) (for SpinCore *PulseBlaster* control) .

Also, for *Pulsebox* card control, accesses [TVicPort.dll](#) and *TVicPort.sys* (but not needed in reference list) .

Project files:

PulseGenerator.cs
 Abstract base class for all pulse generator devices.
PbxControllerCard.cs
 Controller for NOAA *pulsebox* card.
PulseBlaster.cs
 Controller for SpinCore *PulseBlaster*.
PortIO_TVicPort.cs
 Wrapper class for [TVicPort](#) library calls to access *pulsebox* card (32-bit systems).

Contains classes to control pulse generating devices.

AD9959EvalBdUsbK.dll

Friday, February 12, 2016
2:48 PM

References:

DACarter.Utilities.dll
DACarter.PopUtilities.dll

Project Files:

AD9959EvalBdUsbK.cs
LibUsbK.cs
UsbHelper_LibUsbK.cs

Class to access the Analog Devices AD9959 Evaluation Board
via USB with the board jumpered to "PC" mode.

Uses LibUsbK library

<http://sourceforge.net/projects/libusbk/>

Do NOT need to have libusbK.dll referenced,
BUT must include libusbk.cs in project

See AD9959 [Installation Notes](#).

MCPowermeter.dll

Friday, February 12, 2016
2:48 PM

References library mcl_pm64.dll downloaded from Mini-Circuits.

Project File: MCPowerMeter.cs

Control software for Mini-Circuits USB Smart Power Sensor PWR Series

http://www.minicircuits.com/softwaredownload/PM_Programming_Manual.pdf

ZGraphDac.dll

Friday, February 12, 2016
6:14 PM

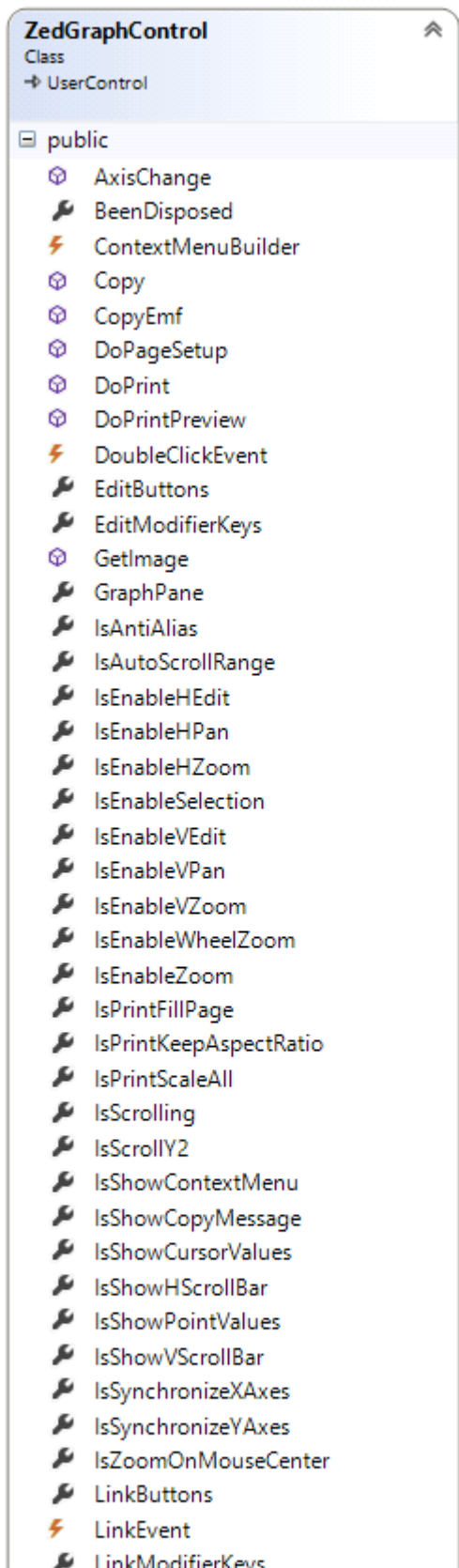
This project takes source code from the [ZedGraph](#) graphics library, adds customized methods and compiles it into the *ZGraphDac* library.





















































The customization was done to the *ZedGraphControl* class and saved in the file renamed to *ZedGraphControl_dac.cs* .

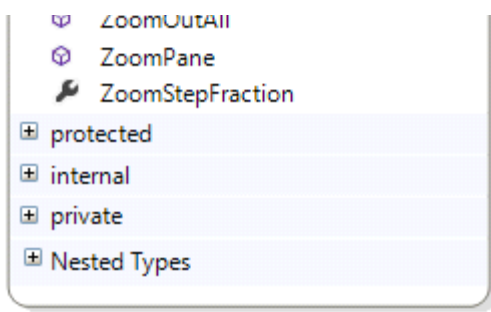
See the [class diagram](#) for a list of all public members of *ZedGraphControl*.

Class Diagram

Thursday, February 18, 2016
4:11 PM



-  LinkButtons
-  LinkEvent
-  LinkModifierKeys
-  MasterPane
-  MouseDown
-  MouseDownEvent
-  MouseMoveEvent
-  MouseUp
-  MouseUpEvent
-  OkToRedraw
-  PanButtons
-  PanButtons2
-  PanModifierKeys
-  PanModifierKeys2
-  PointDateFormat
-  PointEditEvent
-  PointValueEvent
-  PointValueFormat
-  PrintDocument
-  RestoreScale
-  SaveAs (+ 1 overload)
-  SaveAsBitmap
-  SaveAsEmf
-  SaveFileDialog
-  ScrollDoneEvent
-  ScrollEvent
-  ScrollGrace
-  ScrollMaxX
-  ScrollMaxY
-  ScrollMaxY2
-  ScrollMinX
-  ScrollMinY
-  ScrollMinY2
-  ScrollProgressEvent
-  SelectAppendModifierKeys
-  SelectButtons
-  Selection
-  SelectModifierKeys
-  SetScrollRangeFromData
-  UseBitmapCopy
-  Y2ScrollRangeList
-  YScrollRangeList
-  ZedGraphControl
-  ZoomButtons
-  ZoomButtons2
-  ZoomEvent
-  ZoomModifierKeys
-  ZoomModifierKeys2
-  ZoomOut
-  ZoomOutAll
-  ZoomPane
-  ZoomStepFraction



Other Libraries and Programs

Thursday, February 11, 2016

3:41 PM

Numerical Methods Library

Friday, February 12, 2016
3:18 PM

NumericalMethodsLibrary.dll is referenced by DACarter.Utilities.Maths.dll project.

We used the numerical methods library found at

<https://numerical.codeplex.com/>

download source code, numerical-73267.zip, at

<http://download-codeplex.sec.s-msft.com/Download/SourceControlFileDownload.ashx?ProjectName=numerical&changeSetId=73267>

The DLL, NumericalMethodsLibrary.DLL

from NumericalMethods_v.9.3_install.zip

<https://numerical.codeplex.com/downloads/get/95024>

<https://numerical.codeplex.com/SourceControl/latest#NumericalMethods2/NumericalMethodsLibrary/Regression/PolynomialLeastSquareFit.cs>

MathNet Library

Thursday, February 11, 2016
3:41 PM

MathNet.Iridium.dll

The Math.Net Iridium math library is an open source project. It has since been merged into Math.Net Numerics

(see <http://numerics.mathdotnet.com/>).

The downloaded library was precompiled. A help file is available in the Visual Studio MathNet.Iridium-2008.8.16.470 project folder.

We have not tried to update the library to the current Math.Net Numerics library and we do not know if they are compatible without source code changes to POPN.

ZedGraph Graphics Library

Thursday, February 11, 2016
3:42 PM

ZedGraph is a class library, user control, and web control for .net, written in C#, for drawing 2D Line, Bar, and Pie Charts. It features full, detailed customization capabilities, but most options have defaults for ease of use.

Pasted from <<https://sourceforge.net/projects/zedgraph/>>

<http://zedgraph.sourceforge.net/samples.html>

<https://sourceforge.net/projects/zedgraph/>

```
//=====
//ZedGraph Class Library - A Flexible Line Graph/Bar Graph Library in C#
//Copyright © 2004 John Champion
//
//This library is free software; you can redistribute it and/or
//modify it under the terms of the GNU Lesser General Public
//License as published by the Free Software Foundation; either
//version 2.1 of the License, or (at your option) any later version.
//
//This library is distributed in the hope that it will be useful,
//but WITHOUT ANY WARRANTY; without even the implied warranty of
//MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
//Lesser General Public License for more details.
//
//You should have received a copy of the GNU Lesser General Public
//License along with this library; if not, write to the Free Software
//Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
//=====
```

The source code from *ZedGraph* was integrated into the NOAA library project [ZGraphDac.dll](#) where additional customized methods were added.

Progressbar.dll

Friday, February 12, 2016
4:04 PM

The progress bar that is used on the POPN Control Panel is created by the code in project *XpProgressBar*.

Fx2loader.exe

Friday, February 12, 2016
3:26 PM

Also requires the firmware file `AD9959_FW.hex` to be in the executable folder.

64-bit driver did not exist for AD9959 Evaluation Board. Obtained firmware from Analog Devices and wrote code for POPN to configure the device.

In *POP4Service* project, file *LoadDDSFirmware.cs* defines class that uses fx2loader to download firmware to AD9959 so it will renumerate to the proper device.

Method *CheckDDSDeviceDriver()* in *POPNDwellWorker* class calls *LoadDDSFirmware.Run()*.

Procedure:

- Run LibUsbK inf wizard and create libusbk device driver for default device, vid:0456 pid:ee06

 - This needs to be done once per machine.

- Run: `fx2loader -v 0456:ee06 AD9959_FW.hex`

 - If device has not renumerated, it will configure as AD9959 device (0456:ee07)

 - If device is already configured, nothing will happen.

- Run inf wizard on new device (one time) so we have libusbk driver for AD9959.

- fx2loader must be run every time DDS powers up.

- Run fx2loader at the beginning of POPN startup.

<https://github.com/makestuff/libfx2loader/blob/master/README>

<https://github.com/makestuff/fx2loader>

C:\Cypress\fx2loader\libfx2loader-20110913.tar\libfx2loader-20110913\win32\dbg\fx2loader.exe

C:\Cypress\fx2loader.win.x64.20130618\win.x64\dbg\fx2loader.exe

TVicPort.dll

Friday, February 19, 2016
2:34 PM

The library is used to access device port I/O. For POPN this is needed to control the NOAA pulsebox card.

This library is accessed by class *PbxControllerCard* in the [PulseGenDevice](#) project in *DACarter.NOAA.Hardware* solution.

See <http://www.entechtaiwan.com/dev/port/index.shtm> .

The project does not need to reference or have a local copy of *TVicPort.dll*. However, the installation program *TVicPortInstall41.exe* must be run on the computer. This will install
C:\Windows\System\TVicPort.dll and
C:\Windows\System32\Drivers\TVicPort.sys .

I *think* this will only work on 32-bit operating systems, but I have not tried it on 64-bit Windows.
(Support for 64-bit WinXP is claimed.)
Running the install program on 64-bit Windows 10 seems to create the files
C:\Windows\System\TVicPort.dll and
C:\Windows\System32\Drivers\TVicPort64.sys .

SpinAPI.NET.dll

Friday, February 19, 2016
4:25 PM

SpinAPI.NET is a Visual Studio project which creates the library SpinAPI.NET.dll. The project and its source are provided by SpinCore Technologies, with permission to freely alter and redistribute.

<http://www.spincore.com>

The library contains the SpinAPI class, which is used to access and control the SpinCore PulseBlaster device.

The SpinAPI.NET.dll library accesses the SpinAPI.dll or SpinAPI64.dll libraries. These latter libraries are created in \Windows\System32 by the SpinCore installation programs

SpinCore_API_yyyymmdd.exe (for 32-bit OS)

SpinCore_API_yyyymmdd_x86_64.exe (for 64-bit OS)

Where yyyymmdd is different for each update.

The *SpinAPI.dll* libraries do not need to be in the reference list for the *SpinAPI.NET* project, but the *SpinAPI.NET.dll* library must be referenced by the project that calls it ([PulseGenDevice](#)).

See SpinCore [installation notes](#).

POPN Installation Procedure

Tuesday, January 5, 2016
3:03 PM

1. Copy POPN4 Install folder to executable c:_FMCW\POPN4 folder
2. Copy IPP 32- or 64-bit DLLs to executable folder
3. Install DAQ libraries

Device Installation

Friday, February 19, 2016
6:15 PM

DAQ Installation

Friday, February 19, 2016
6:16 PM

To install [MCC DAQ](#) device driver libraries:

Run mccdaq.2.6.exe program.

Installs DAQ example source code in "public\documents\Measurement Computing"

Applications, libraries, documents installed in "program files (x86)\Measurement Computing"

(these folders are specified in the registry, e.g. HKLM\Software\Universal Library and others)

Reboot before running any C# programs

Plug in the USB device; it shows up either as "DAS Component -> USB 2523" or DaqBoard3005

Select "Update Driver" to change which device type. Select from "choose from list of devices".

With libraries and cfg file locations specified in registry, do not need any files in executable path;

In Visual Studio need only to reference MccDaq.dll

(in program files (x86)\Measurement Computing\DAQ).

This dll can be found in the .NET tab of the "Add Reference" dialog.

BUT, the version of MccDaq.dll on the runtime computer must match the version that POPN was compiled with, even though MccDaq.dll is not distributed with POPN.

So if the runtime system is upgraded by running a newer version of mccdaq.exe, the POPN program must also be recompiled with the newer MCC version.

Or, you need to run the proper version of mccdaq.exe (which installs drivers and library dll) that matches the version the POPN was built with.

As of February 2016, POPN4 was compiled with MCC version 2.6 .

To install [IOTech DAQ](#) drivers:

Will work only with 32-bit POPN4

Run daqcom2setup.exe (version 2.2.1.1 displayed) installs program support.

Run daqviewsetup.exe (version 9.1.36 displayed) installs drivers.

Build POPN4 DAQDevice.dll routines with reference to IOTech daqcomlib.dll

Select version 2.401 from COM library in Add Reference.

Select device drivers 5.0.0.11 for USB2 Loader Device and DAQBoard/3005USB

SpinCore Installation

Friday, February 19, 2016
6:32 PM

The [SpinAPI.NET.dll](#) library accesses the SpinAPI.dll or SpinAPI64.dll libraries. These latter libraries are created in \Windows\System32 by the SpinCore installation programs

SpinCore_API_yyyymmdd.exe (for 32-bit OS)

SpinCore_API_yyyymmdd_x86_64.exe (for 64-bit OS)

Where yyyymmdd is different for each update.

To Install SpinCore API libraries:

1) Disconnect PulseBlaster card then uninstall previous versions of SpinAPI.

To uninstall, run c:\spincore\spinapi\uninsxxx.exe

2) Run SpinAPI installer exe as administrator: SpinCore_API_xxxxxx.exe (32-bit)

SpinCore_API_xxxxxx_x86_64.exe (64-bit)

3) Install PulseBlaster card and turn on PC

4) For WinXP Found New Hardware:

Choose install from specified location:

c:\spincore\spinapi\drivers\pci

5) Will show up as multifunction adapter in device list.

AD9959 DDS Installation

Friday, February 19, 2016
6:43 PM

Disconnect DDS device.

Run *AD9959_Setup1.0.exe* in *POPN Extras\AD9959*

Then install USB drivers:

For POPN4:

(Files in *POPN Extras\LibUsbK*)

To install LibUsbK drivers for AD9959 DDS

Run *libusbk-inf-wizard.exe*

Select device 0456:ee06 or 0456:ee07

Select libusbK drivers

Program [fx2loader.exe](#) and

the firmware file *AD9959_FW.hex*

must be in executable folder.

For POPN3:

(Files in *POPN Extras\LibUsb2.2.8*).

<http://sourceforge.net/projects/libusbdotnet/>

LibUsb-win32 is installed at same time.

Plug in AD9959 (dismiss Found Hardware dialog)

run inf-wizard (10/29/2010 2.2.8.104)

if AD9959 is in list, select it, else

(create new) to create *.inf file for specific device:

VID = 0x456

PID = 0xEE07

specify folder for output

select title for files (e.g. DDS_USB_20110818)

select name for device (e.g. AD9959)

next ->

creates folder (DDS_USB_20110818) in output folder

which contains drivers, etc.

click Install Driver

When plug in DDS USB first time -> found new hardware

Should let Windows find driver, else

Select install from specified folder and choose

folder with the above driver files

or this POPN Install folder (?).

MC Power Meter Installation

Wednesday, February 24, 2016
12:43 AM

Library file mcl_pm64.dll (downloaded from minicircuits.com) must be in POPN executable folder.

Upgrade PC to Win10

Wednesday, February 24, 2016
12:51 AM

NOTE: To upgrade system to Win10, need another admin login besides "administrator", because "administrator" is disabled in Win10.
Once logged in, enable "administrator" via users settings.

Introduction to WCF

Saturday, February 13, 2016
6:10 PM

One of the main goals for the design of the POPN program was to have the main program run as a Windows service, independently of the Control Panel application. Therefore we needed a way to communicate between these two processes. I examined various ways to do this, but the main recommendation at the time was to use the new Windows Communication Foundation (WCF) protocol.

As Microsoft explains:

Windows Communication Foundation (WCF) is a framework for building service-oriented applications. Using WCF, you can send data as asynchronous messages from one service endpoint to another. A service endpoint can be part of a continuously available service hosted by IIS, or it can be a service hosted in an application. An endpoint can be a client of a service that requests data from a service endpoint. The messages can be as simple as a single character or word sent as XML, or as complex as a stream of binary data.

...

While creating such applications was possible prior to the existence of WCF, WCF makes the development of endpoints easier than ever. In summary, WCF is designed to offer a manageable approach to creating Web services and Web service clients.

Pasted from <<https://msdn.microsoft.com/en-us/library/ms731082%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396>>

WCF turns out to be quite complex and may have been overkill for the use intended in POPN.

Communication is handled by the classes in the *PopCommunication* project of *POPN* and in the library *DACarter.ClientServer.dll*.

The POPN service is set up as a WCF server and the *ControlPanel* program is a client. In general, the client calls methods implemented on the server, but the server can call "callback" methods implemented on the client.

POPCommunicator Class Diagram

Saturday, February 13, 2016
5:11 PM

WCF Example

Saturday, February 13, 2016
6:33 PM

The following are instructions on how to use the DACarter.ClientServer library to implement WCF to communicate between two processes. This procedure was followed to create the host and client objects in PopCommunication project in POPN.

```
//
// DacClientServer DacClientServer.cs
//
// This file contains helper classes for creating
// WCF client-server duplex messaging .
//
// Example:
// Server called MyServer with method ServerMethod
// Client called MyClient with callback method CallbackMethod
//
// *****
// COMMON INTERFACE
//
// First define client and server interfaces.
// IMyServer must include Subscribe and Unsubscribe methods
//   in addition to custom methods.
//   Subscribe and Unsubscribe are already implemented in server base class.
// ICallback methods must be "oneway", i.e. no return value or out parameters.
/*
    public interface ICallback {
        [OperationContract(IsOneWay = true)]
        void CallbackMethod(MyData data);

        [OperationContract(IsOneWay = true)]
        void AnotherCallback(MyData data);
    }

    [ServiceContract(CallbackContract = typeof(ICallback), SessionMode = SessionMode.Required)]
    public interface IMyServer {

        [OperationContract]
        string ServerMethod(string command);

        [OperationContract]
        bool Subscribe();

        [OperationContract]
        bool Unsubscribe();
    }
*/
// And define data objects that are transferred, for example
```

```

/*
    [Serializable]
    public struct MyData {
        public string Message;
        public int Count;
    }
*/
// The above assembly must be referenced by both client and server.
//
// *****
// CLIENT
//
// In the client, implement the callback methods and create client class.
/*
    public class CallbackImpl : ICallback {
        public void CallbackMethod(MyData data) {
            // do something
        }
        public void AnotherCallback(MyData data) {
            // do something
        }
    }

    public class MyClient<I,C> : DacClientBase<I,C> where C : new() {

        private IMyServer _proxy;

        // get reference to MyServer methods in ctor
        public MyClient(TransportMethod transport) : base( transport) {
            _proxy = GetProxy() as IMyServer;
        }
        public void SomeMethod() {
            // call server method at some point:
            string returnValue = _proxy.ServerMethod(string arg);
            // ...
        }
    }
*/
// In the client executable program:
// Note: client contains public member CallbackObject
// that references the callback class implementation object.
/*
    static void Main(string[] args) {
        // ...
        MyClient<IMyServer, CallbackImpl> client = new MyClient<IMyServer, CallbackImpl>
(TransportMethod.NamedPipes);
        client.SomeMethod();
        // ...
    }
*/
//
// *****
// SERVER

```

```

//
// Define your server class, implement the server method,
// and make callbacks to clients:
/*
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
public class MyServer<IC> : DacServerBase<IC>, IMyServer {
    // ...
    public string ServerMethod(string command) {
        // do your thing ...
        // callback:
        CallbackToClients("CallbackMethod", data);
        // ...
    }
}
*/
// *****
// HOST
//
// The host for the server can be a windows or console app or a service.
// Create DacServerHost object.
// Server is active as long as DacServerHost object lives.
// Must call Dispose() at the end of all paths
/*
    TransportMethod transport = TransportMethod.NamedPipes;
    DacServerHost<MyServer<ICallback>, IMyServer> dacHost = null;
    try {
        dacHost = new DacServerHost<MyServer<ICallback>, IMyServer>(transport);
    }
    catch (Exception e) {
        if (dacHost != null) {
            dacHost.Dispose();
        }
        return;
    }
    if (dacHost.IsRunning) {
        // Do your thing...
    }
    // Before terminating:
    dacHost.Dispose();
*/
*/

```

Outline of the WCF Implementation

Monday, February 15, 2016
12:21 PM

The communication protocol is used in various situations.

1. Sending commands from the control panel to the POPN service.
2. Sending status information from POPN service to the control panel.
3. Sending data from POPN service for plotting in the control panel.
4. Sending parameters used by the POPN service to the control panel.

Examples (see the [POPCommunicator Class Diagram](#)):

1. Sending the Start Command from the control panel.
 - a. *buttonGo_Click()* in *POP4MainForm* calls the *SendCommand* method of the *POPCommunicator* client.
 - b. That *POPCommunicator.SendCommand* calls the *POPCommClient.SendCommand* method.
 - c. *POPCommClient* calls *IPOPCommServer.NewCommand* method which executes on the *POPCommServer* and calls *POPCommunicator.FireCommandReceivedEvent*.
 - d. This calls the *CommandReceived* event handler defined in *POP4Service*:
`communicator.CommandReceived += _dwellWorker.CommandReceived;`
 - e. The *CommandReceived* method in *POP4Service.PopNDwellWorker* adds the command to the command queue.
 - f. The command queue is read by the *CheckCommand* method of *PopNDwellWorker* (in the *WaitForGo* method, for example).
2. Sending a status message to the control panel.
 - a. Create a *PopStatusMessage* object initialized with the text message, running status, etc. that you want to send.
 - b. Send the status message object by calling *POPCommunicator.UpdateStatus*.
 - c. This *POPCommunicator.UpdateStatus* calls the *UpdateStatus* method of the *POPCommServer* object stored in the *POPCommServerHost* object.
 - d. The *POPCommServer.UpdateStatus* method calls
`CallbackToClients("OnStatusUpdate", status)`
 - e. *CallbackToClients* calls the callback method that has the name "OnStatusUpdate".
 - f. The *OnStatusUpdate* calls the *POPCommunicator* (client) method *FireStatusUpdatedEvent*.
 - g. *FireStatusUpdatedEvent* calls the event handler defined in *POP4MainForm*:
`_communicator.StatusUpdated += OnStatusUpdated;`
 - h. *POP4MainForm.OnStatusUpdated* puts the *PopStatusMessage* object on the status queue.
 - i. The status queue is checked every tick of the *statusTimer*.
3. Sending data to be plotted by the control panel application.
 - a. There are two techniques used.
 - i. Sending an array of data via the *POPCommunicator* protocol.
 - ii. Due to a limit on how large an array can be sent that way, we have switched some plots to get data from shared memory, described here for time series data.
 - b. The *PlotDopplerTS* method in *POP4MainForm* creates a new memory mapped file to hold the data.
 - c. The *PlotDopplerTS* method calls the *RequestDopplerTS* method of the *POPCommunicator* client.
 - d. The command is transmitted to *PopNDwellWorker* in the *POP4Service* in a manner similar to sending commands from the control panel (Item 1) above.
 - e. The *DopplerTSRequested* method of *PopNDwellWorker* accesses the existing memory mapped file and transfers the requested data set to the memory mapped file.

- f. In *POPN4MainForm*, when the call to *RequestDopplerTS* returns, the data in the memory mapped file is plotted.
- 4. Sending parameter info from the service to the control panel.
 - a. When *POPN4MainForm.UpdateStatus* detects that the dwell is finished and data is available, *RequestParameters* method of the *POPCommunicator* client is called.
 - b. A communications protocol similar to item 1 above is followed with an out parameter of type *PopParameters* being returned to *POPN4MainForm*.

Parameter File

Wednesday, February 17, 2016
2:09 PM

Parameters that are set by the user and that control the operation of the POPN program are stored in files with a *.parx* extension.

In the POPN code these parameters are stored in an object of type *PopParameters*, defined in the *DACarter.PopUtilites.dll* library.

The *parx* file is basically an image of the *PopParameters* object. The *PopParameters* object is transferred to the *parx* file using .NET serialization by the *PopParameters.WriteToFile* method:

```
public void WriteToFile(string filePath) {  
    StreamWriter ParFileWriter = new StreamWriter(filePath);  
    XmlSerializer ParFileSerializer =  
        new XmlSerializer(typeof(PopParameters));  
    ParFileSerializer.Serialize(ParFileWriter, this);  
    ParFileWriter.Close();  
}
```

The resulting parameter file is a text file in XML format. Each field of the *PopParameters* object becomes an item in the XML file. Parameters can be added or removed by redefining the fields of the *PopParameters* class. There is never a conflict with one version of the *POPN* program trying to read an older (or newer) version of the parameter file. If *POPN* reads an XML item in the file that is not defined in the *PopParameters* class, it ignores it. If the file is missing an item that is defined in the class, it is assigned a default value.

The values of parameters are normally set through the parameter setup screen of the *POPN4ControlPanel* program. However, since it is a text file, the *parx* parameter file can also be edited manually to change the value of parameters.

Structure of the Parameter File

Wednesday, February 17, 2016
3:13 PM

The structure of the parameter file is defined by the public fields of the *PopParameters* class in the *DACarter.PopUtilities* library.

The public fields of *PopParameters* are:

```
public double PopParametersVersion = 20151208;
public string Source;
public ArrayDimensions ArrayDim;
public SystemParameters SystemPar;
public MeltingLayerParameters MeltingLayerPar;
public ReplayMode ReplayPar;
public ModeExcludeIntervalsStruct ExcludeMomentIntervals;
public SignalPeakSearchRangeStruct SignalPeakSearchRange;
public DebugOptions Debug;
```

The various data types (all structs) are defined in *PopParameters.cs* . Some are described below.

The first field, *PopParametersVersion*, by custom should be set to the current date when a modification is made to the parameter types.

The *Source* string contains the name of the file or program that created this parameter set.

The main parameter structure is *SystemParameters*, which is defined as

```
public struct SystemParameters    {
    public string StationName;      // station name
    public double Latitude, Longitude; // N latitude, E longitude
    public int MinutesToUT;        // # minutes add to system time to get UT
    public int Altitude;           // altitude above sea level, meters
    public int NumberOfRadars;     // number of radars at this station
    public RadarParameters RadarPar; // radar id structure for each radar
}
```

RadarParameters type is defined as

```
public struct RadarParameters {
    private int _numOtherInstruments;
    public string RadarName;      // name of radar
    public int RadarID;           // ID code number for radar
    public TypeOfRadar RadarType; // Pulsed, FMCW, etc.
    public double TxFreqMHz;      // tx freq in Mhz
    public double AntSpacingM;    // spacing between centers of spaced antennas (m)
    public double ASubH;
    public double MaxTxDutyCycle; // max duty cycle
    public int MaxTxLengthUsec;   // max Tx pulse length (microsec)
    public double MinIppUsec;     // minimum IPP in usec
    public bool TxIsOn;           // tx pulse on or off
}
```



```

public BeamPosition[] BeamSequence; // array of chosen beam positions [MAXBM]
public BeamParameters[] BeamParSet; // array of beam position parameter sets [MAXBMPAR]
public Direction[] BeamDirections; // array of allowable directions [MAXDIR]
public PbxConstants PBConstants; // pulse box constants for this radar
public RxBwParameters[] RxBw; // matched pulsewidths (nsec) for rx bandwidth [MAXBW]
                                // plus total extra delay for each rx bw

public ProcessingParameters ProcPar;
public FmCwParameters[] FmCwParSet;
public PowerMeterParameters PowMeterPar;

public int NumOtherInstruments;
public int[] OtherInstrumentCodes;
}

```

The *ArrayDimensions* type defines the (fixed) dimension sizes of various arrays in the parameter file:

```

public struct ArrayDimensions {
    public int MAXBEAMS; // default for backwards compatibility with POP4 = 10
    public int MAXBEAMPAR; // default = 4
    public int MAXDIRECTIONS; // default = 9
    public int MAXBW; // default = 4
    public int MAXCNSMODES; // default = 2
    public int MAXOUTPUTFILES;
    public int MAXRXID;
}

```

Creating a New Parameter

Wednesday, February 17, 2016
3:01 PM

As an example, let's say we want to create a new parameter called *OptionX* that is either on or off.

First, define a field in the *PopParameters* class call *bool OptionX*. It can be a public field of *PopParameters* or a field in one of the subtypes. For example, if it is a processing option we could define it inside the *ProcessingParameters* struct.

Next, we must modify the *PopParameters.Equals()* method to test the *OptionX* field of two parameter sets for equality.

Then, we probably want to be able to read and write this parameter from the parameter setup screen in the *Control Panel* program. So, in Visual Studio, in the *POP4ControlPanel* project of the *POP4* solution, open *PopNSetup3.cs* in Designer view. Add a *CheckBox* control called *checkBoxOptionX* on an appropriate tab page.

Now add code to read and write the parameter file. So right click on the *PopNSetup3.cs* designer view and select "view code". To write the parameter to the file, look at the method *SaveAllTabPage()* and select the call for the page that contained our new checkbox -- for example, *SaveProcessingPage()*. In that method add a line similar to the following:

```
_parameters.SystemPar.RadarPar.ProcPar.OptionX =  
checkBoxOptionX.Checked;
```

Now the user-selected value for *OptionX* is in the *PopParameters* object and will be saved in the *parx* file.

To display the value of the parameter stored in the parameter file, go to the method *FillProcessingPage* (methods for the other pages are found in method *FillParameterScreens*). Insert this:

```
checkBoxOptionX.Checked =  
_parameters.SystemPar.RadarPar.ProcPar.OptionX;
```

When adding a new parameter control to a tab page on the setup screen, if you want to modify other controls based on input to the new control or if you want to validate the entry, you will want to add code for events such as the *TextChanged* event in the Designer. See examples of this in the code.

Replay Mode:

The next issue to consider is what to do with parameters in the parameter file when reading recorded data (replay mode). Some parameters are determined by data in the data file. Examples would include radar parameters such as IPP, number of samples taken, system info, etc. These parameters from the data file must not be overwritten by the values that are in the *parx* file. Other parameters tell POPN how to process the data from the data file. Examples of these processing parameters include spectral and cross-correlation settings, window functions and filtering to use, etc. These parameter values must be copied from the *parx* file to the parameter object of the recorded data.

To set the proper parameter object in replay mode, go to method *GetReplayData()* in *PopNDwellWorker.cs*. Here the data and parameters from the data file arrive in a *PopDataPackage3* object. In this method various processing parameters from the *parx* file (in *_parameters*) are copied into the *PopDataPackage3* object. For example:

```
samplpDataPackage.Parameters.MeltingLayerPar =
```

```
_parameters.MeltingLayerPar;
```

NOTE 1: There are some radar system parameters that do not have an equivalent item in the POP data file format, such as `_parameters.SystemPar.RadarPar.RadarType` . These values are read from the *parx* file used for replay and must be set to a value that matches the data that was recorded or else the correct processing may not take place.

NOTE 2: Most of the processing parameters are copied over to the data parameter set in *PopNDwellworker.GetReplayData()* as described above, but you will find a few that are set in *PopNReplay.Pop5ToPopN()* . I have not yet taken the time to verify if these can be moved to *GetReplayData* or whether they need to be set earlier in *Pop5ToPopN* (e.g. in order to allocate arrays of correct sizes).

Setting Up Version Control

Wednesday, February 17, 2016
2:39 PM

Using *Git* for version control.

"**Git** is a widely used source code management system for software development. It is a distributed revision control system... As with most other distributed version control systems, and unlike most client-server systems, every Git working directory is a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server."

Pasted from <[https://en.wikipedia.org/wiki/Git_\(software\)](https://en.wikipedia.org/wiki/Git_(software))>

Using Bitbucket for the remote repository for the source code (free for up to 5 team members).
<http://bitbucket.org>

Documentation for BitBucket:

<https://confluence.atlassian.com/bitbucket/bitbucket-cloud-documentation-home-221448814.html>

The source repository is on the local Visual Studio machine, but on command it is synchronized to the remote repository on Bitbucket.

The repository for the POPN solution is named POPN_NOAA_PSD
https://david_carter@bitbucket.org/david_carter/popn_noaa_psd.git

It is accessed through web page

https://bitbucket.org/david_carter/popn_noaa_psd

Login:

User Name: david_carter

Password: l3m0n!m3

GitExtensions installed into Visual Studio 2013
from <https://sourceforge.net/projects/gitextensions/>

David.A.Carter

gitext@gocarter.com

Installed kdiff3 to be diff tool to use in GitExtensions

C:/Program Files/KDiff3/kdiff3.exe

<http://kdiff3.sourceforge.net/>

After installing and setting up GitExtensions in Visual Studio,

Display GitExt toolbar in VS.

Can also run GitExtensions in desktop app

C:\Program Files (x86)\GitExtensions\GitExtensions.exe

Created new local repository in POPN4 solution folder

C:\Users\Dave\Documents\Visual Studio 2013\Projects\POPN4

and added files to it.

New users (team members) will probably need to clone the repository from the Bitbucket site to their local computer.

Select *Clone Repository* from GitExt menu

and enter http address of POPN_NOAA_PSD (see above).

Alternative interface:

Use the SourceTree program

<https://www.sourcetreeapp.com/download/>

Select Connect to Bitbucket and

log in with the Bitbucket user and password

I found this useful for creating a new repository outside of Visual Studio.

The POPN solution has the remote repository on Bitbucket called POPN_NOAA_PSD.

I had created local repositories for the 7 library projects under DACarter folder.

But I had not pushed these to the remote site.

So I created 7 new repositories on Bitbucket for DACarter library projects.

Locally, using File Explorer context menu, "pushed" each subfolder to the proper remote repository.

Note that there is a remote repository called DACarter-Libraries, which was created in an attempt to combine all the library projects into one repository. I didn't know what I was doing well enough to make that work. So you can ignore that repository.

Saving Versions

Tuesday, February 23, 2016
10:38 AM

Work flow for backing up file versions:

1. Bring up "Commit" window
 - a. Click *Commit* on GitExt toolbar in Visual Studio, or
 - b. Click *Commit* in GitExt menu in VS, or
 - c. Right click the folder in File Explorer to bring up GitExt context menu.
2. Upper left panel shows new or changed files.
3. Select files to commit to repository and click "Stage" button.
4. Staged files move to lower left panel.
5. Enter a version label or short message in lower right panel.
6. Click "commit" button to update local repository only.
7. Click "commit and push" to update local and remote repository.

To view the repository:

1. Click "Browse" in Gitext VS toolbar or menu, or Explorer context menu.
2. Upper panel shows version tree of the project.
3. Click on a version to view.
4. Select "File Tree" tab in lower left panel to see list of files.
5. Select "Diff" tab to see changes in files that were modified for this version.

To create a new repository from a Visual Studio project:

1. Open project in Visual Studio.
2. The GitExt menu item "create new repository" does not seem to work.
3. If you click on "browse repository" you get message that this is not a Git repository.
4. Below this message is an "initialize repository" button. Click it.
5. Brings up "create new repository" dialog.
6. Select personal repository type.
7. Make sure that the directory displayed is the project's main folder.
8. Click "create".
9. Click "edit .gitignore". This file lists file types not to include in the repository, e.g. Visual Studio binary files, log files, temporary files, etc.
10. You can copy and paste content from another .gitignore file, type in your own list, select "add default ignores", or select "more gitignore patterns" and copy and paste from "Visual Studio gitignore".
11. After you save the gitignore file, click the "commit" beneath the "gitignore" button.
12. Then you get the "commit" window and follow the procedure above for "backing up file versions".
13. If files are listed in the "changes" list that you do not want in the repository, you can right click on the file name and select "add file to .gitignore".
14. "Stage" and "commit" the selected files.
15. If you also want a remote repository, you must create a repository in Bitbucket before you can "push" the changes in the local repository to the remote.

Other Considerations

Tuesday, February 23, 2016
10:57 AM

Git version control allows you to create branches to the code development so that you can make changes to the branch without modifying the main branch. When the branch has been sufficiently tested, it can be merged back into the main branch.

I have not worked with branches much in *Git*, but I think it works like this.

1. Create branch and checkout (now working copy is Branch).
2. Commit revisions (to Branch).
3. To make changes in Master, select Master and checkout. Now working with previous version, without Branch edits. Revisions are committed to Master branch.
4. To merge, checkout Master branch. Select Merge Branches or Merge into current branch. Branch is merged into Master. If changes have been made to Master after branching, merge does not happen if files conflict. Will ask you if you want help eliminating conflicts.
5. After merging, now only have one branch (Master), but can go back to Branch or unmerged Master.

Synchronizing versions among multiple developers is discussed here:

<https://www.atlassian.com/git/tutorials/syncing>

Different types of collaborative workflows are discussed here:

<https://www.atlassian.com/git/tutorials/comparing-workflows>

<http://blog.endpoint.com/2014/05/git-workflows-that-work.html>

For simple, small projects like this probably best just to make sure only one developer at a time is modifying the source code that is in the remote repository and that that person commits and pushes his changes before another person edits the code. Maybe if the new editor creates a new branch for his revisions, it will be easier to identify the latest tested version (Master) from the new updates (Branch).

File Backup

Wednesday, February 24, 2016
7:08 PM

Aside from the *git* version control software, I also used other methods to make backups of the POPN project.

I have used *GoodSync* (<http://www.goodsync.com/> version 8.9.9.9) to make backup copies of files to external drives. One backup was for the executable files to install on or update a radar computer. Another was a backup of all the project files from Visual Studio. Exported job files are in the POPN4 Install Extras folder.

The *.tix files are *GoodSync* job files exported from DAC's PC.
Import these into your *GoodSync*.
Modify the source and destination folders to match your folder structure.

GoodSync copies files and subfolders that have changed from a source folder to a destination folder, excluding files and file types in an excluded file list.

The first job, POPN4 Install 1, copies files needed to run POPN4 from the POPN4 output folder (...\\bin\\debug) to a separate folder (c:\\POPN4 Install).

The second job, POPN4 Install 2, copies files from the c: drive install folder to an install folder on an external removable drive.

The next job, VS Projects, copies all important project files to a backup external drive.

GoodSync is not required and it is not a free program (\$29.95), but I have found it helpful and easy to use.

I also have used online backup software on my personal PC (SOS and Crashplan) so I can retrieve backed up versions of the software for the past few years.

Processing

Tuesday, January 5, 2016
12:50 PM

Setup Parameters:

nRx = 3
nSpec
nPts
TxSweepSampleNPts (nSamples)
TxSweepSampleDcFilter (bool)
TxSweepSampleDcFilter2 (bool)
DCFilter (DopplerDcFilter) (bool)
Xcorr Npts Mult (NXCPtMult)
MaxLag
PolyOrder
Fraction LP Filter
SlopeFitPoints
Lags to Fit
Lags to Interp (total number both sides)
doClutterWavelets (bool)
Adjust Base (bool)
FFT (bool)

Derived parameters:

XCorrNPts = nPts * NXCPtMult
XCorrNAvg = nPts / NXCPtMult
NHts = nSamples/2 + 1
doXCorrFilter = true if FractionLPFilter < 1.0

Algorithm:

```
Acquire data samples (nSamples * nPts * nSpec for each receiver).
Process raw samples:
  If TxSweepSampleDcFilter then remove DC from each set of nSamples for each IPP.
  If TxSweepSampleDcFilter2 then remove DC from each set of nPts for each gate.
  Apply window function to each set of nSamples and do FFT.
  Exchange I and Q for proper Doppler sense unless TX sweep freq offset is negative.
  If TX sweep freq offset < 0, reverse order of FFT pts from 0 to nHts-1.
Process Doppler time series:
  If doClutterWavelets
    Apply D20 wavelet filter to each set of nPts for specified # of heights.
    Transform, clip at specified threshold and specified Doppler range, then inv transform.
  Else
    If DopplerDCFilter then remove DC from each set of nPts for each height.
  If doXCorrFilter
    Take FFT of _nPts time series and convert to power series.
    Excluding FractionLPFilter*nPts points around DC, use H&S method to find noise level.
    For all points above maxNoise, set complex array points to 0.0.
    Do inverse FFT.
  Compute cross- and auto-correlations over XCorrNPts for +/- MaxLag lags at each height.
  Average XCorrNAvg correlations together at each height.
  Compute Doppler nSpec spectra of length nPts at each height for each receiver.
  Window time series, then do FFT and convert to power spectra.
Process spectra and correlations:
  Apply gain filter factors to power spectra as function of height.
  Calculate "moments":
    Process spectra:
      If DC filter was applied, then skip (or interpolate) dc point (3 pts if windowing).
      Compute noise level.
      Remove ground clutter.
      Check for excluded regions and signal search restrictions.
      Calculate signal power, Doppler, and width.
    Process correlations:
      Interpolate LagsToInterp/2 points on each side of 0 in auto- and cross-correlations.
      Fit Gaussian to cross-corr.
      Compute slope of Gaussian at zero lag.
      Fit polynomial to cross-corr using all lags. Find the peak.
      Fit another polynomial using only LagsToFit points around the peak.
      Find base level of cross-corr by smoothing the curve and using lowest point as base level.
      Fit polynomial to auto-correlation and find base level.
```

If adjustBase=true, subtract base level from cross- and auto-correlations.
Find intersection of cross- and auto-correlations (τ_i).
Find lag τ_x where $xcorr(\tau_x)$ equals $autocorr(0)$.
Do cross-corr ratio calculation.
Compute straight line fit to xcorr ratio using SlopeFitPoints # pts.
Process moments:
Consensus processing (only on replay for 1 receiver pulsed systems).
Melting layer processing.