phasor data open ECA platform

Test Harness Training

June 29, 2016

DOE FOA 970
DE-OE-778

TEST HARNESS

openECA

OVERVIEW

Analytics Development is Simplified

Today’s Approach

- “Signal” paradigm
- Use IEEE C37.118
  - Socket management
  - Protocol parsing
  - Exception handling
- Local data buffering to support analytic cycle times
- Local configuration management

Using openECA

- Both standard and custom data objects
- An API (the CAI) that provides
  - Hi-performance pub/sub data access using standard messaging (e.g., Zero MQ)
  - Access to meta data services
  - Local data buffering options
- Starter templates provided
  - C# / F#
  - C++
  - Java
  - Matlab

Value to the Industry

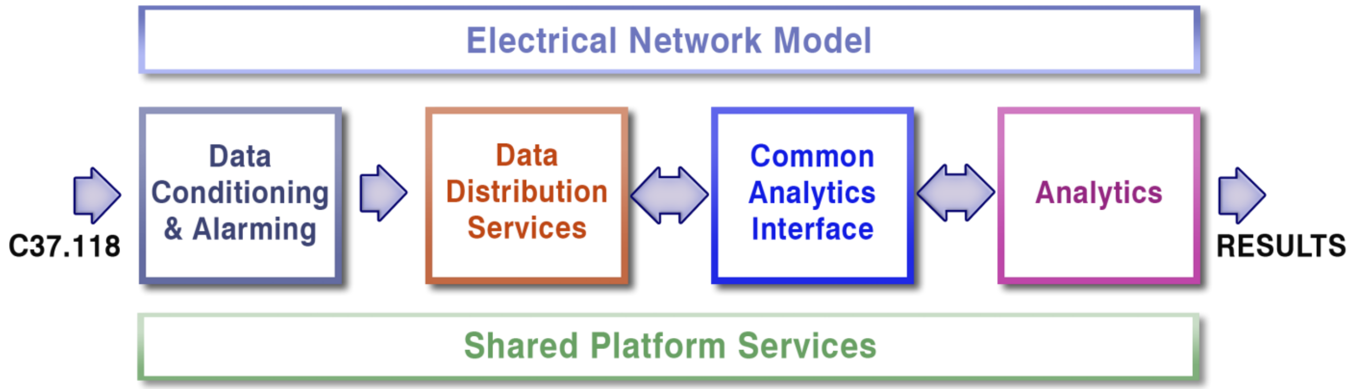
- Lowers cost of addition of new production analytic tools
- Simplified end-to-end configuration and change management
- Improved availability of phasor data with greater visibility of phasor data quality
- Robust scalable solution to support phasor data infrastructure of any size
- Complements current phasor data architecture and supports integration with other data sources such as SCADA

“Out-of-the-Box” Included Analytics

A. Localized Voltage-VAR Controller
B. PMU Instrument Transformer Calibration
C. PMU Synchroscope
D. Real-Time Impendence Calculator
E. Regional Voltage Control
F. Topology Estimator
G. Transmission Line Impedance Calibration
H. Oscillation Detection
I. Oscillation Mode Meter
J. Synchronous Machine Parameter Estimation
K. Acceleration Trend Relay Improvement

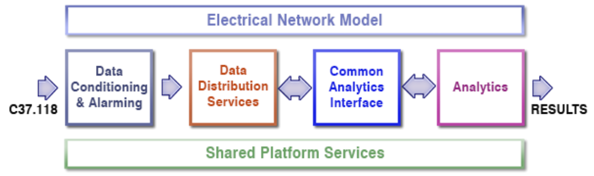
Major Architectural Elements

- Data Integration Services
- Common Analytics Interface (CAI)
- Data Conditioning and Alarming
- Electric Network Model
- Shared Platform Services
- Analytics


Electrical Network Model
C37.118
Data Conditioning & Alarming
Data Distribution Services
Common Analytics Interface
Analytics
RESULTS
Shared Platform Services

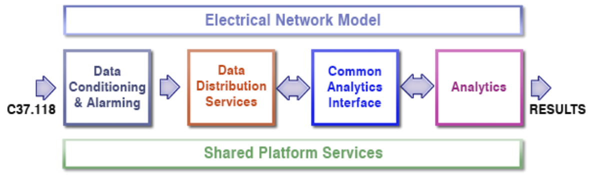
Data Integration Service Summary

- Real-time and historical data acquisition
  - Adapter-driven data providers will include:
    - All common synchrophasor protocols
    - Common RDBM systems
    - OSI-PI and other historians
    - Other protocols, e.g., DNP3, Kafka, COMTRADE
- Device management
  - Automated connectivity
  - Data quality reporting
- Time-series data management

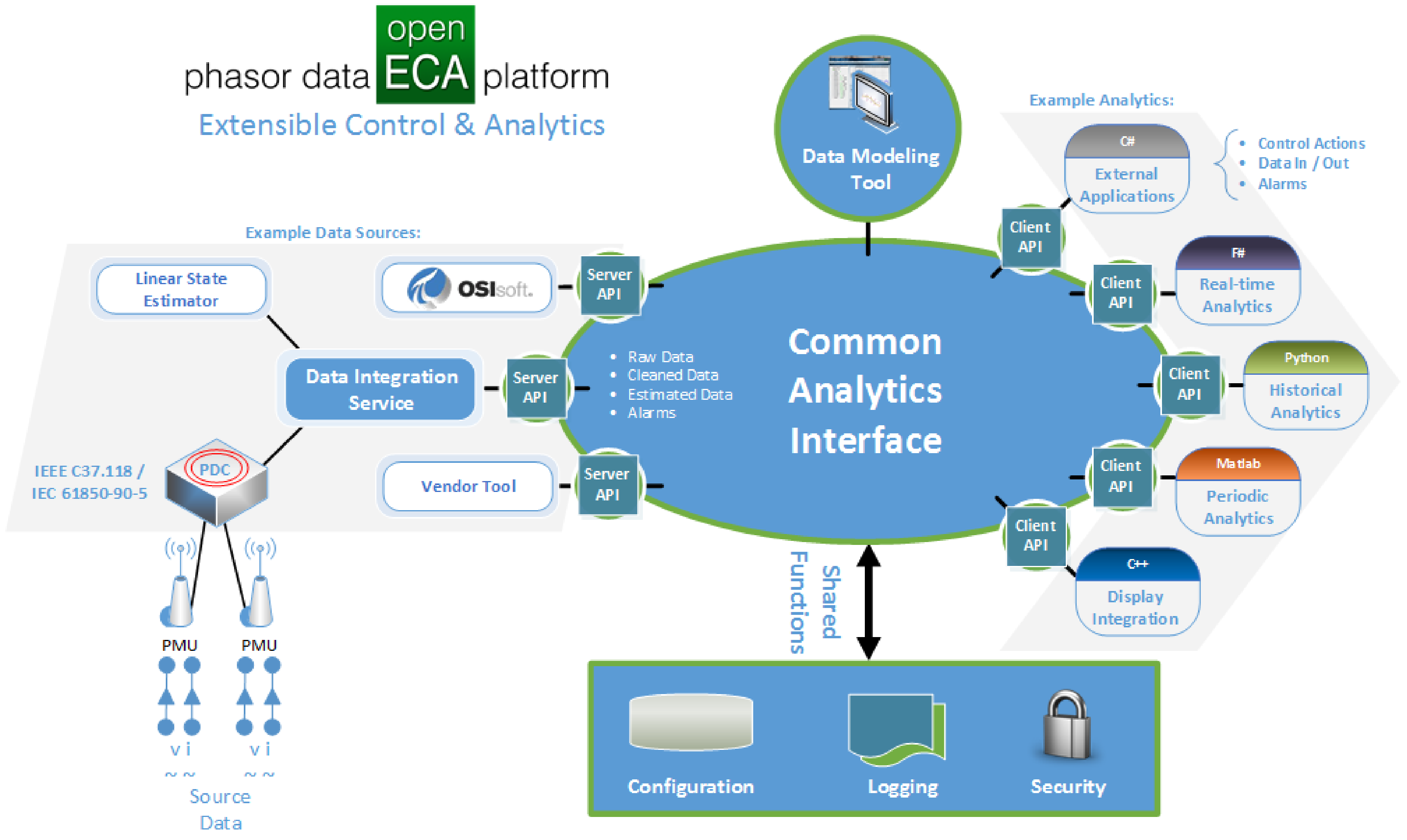

Electrical Network Model
C37.118
Data Conditioning & Alarming
Data Distribution Services
Common Analytics Interface
Analytics
RESULTS
Shared Platform Services

Common Analytics Interface Summary

- Server API (targets .NET)
  - Authorizes client data source connectivity
  - Provisions time-series data and metadata
- Client API (targets multiple platforms)
  - Manages server connectivity
  - Executes data filtering, organization and aggregation over user defined time-intervals
- Data Model Management Tool
  - Defines data filtering, organization and aggregation
  - Trends incoming data sources


Electrical Network Model
C37.118
Data Conditioning & Alarming
Data Distribution Services
Common Analytics Interface
Analytics
RESULTS
Shared Platform Services

System Data Flow Diagram



Product Releases

- Releases will be made available regularly via GitHub – current openECA *Prototype* with operational Analytic Development Test Harness is now available:

https://github.com/GridProtectionAlliance/openECA/releases

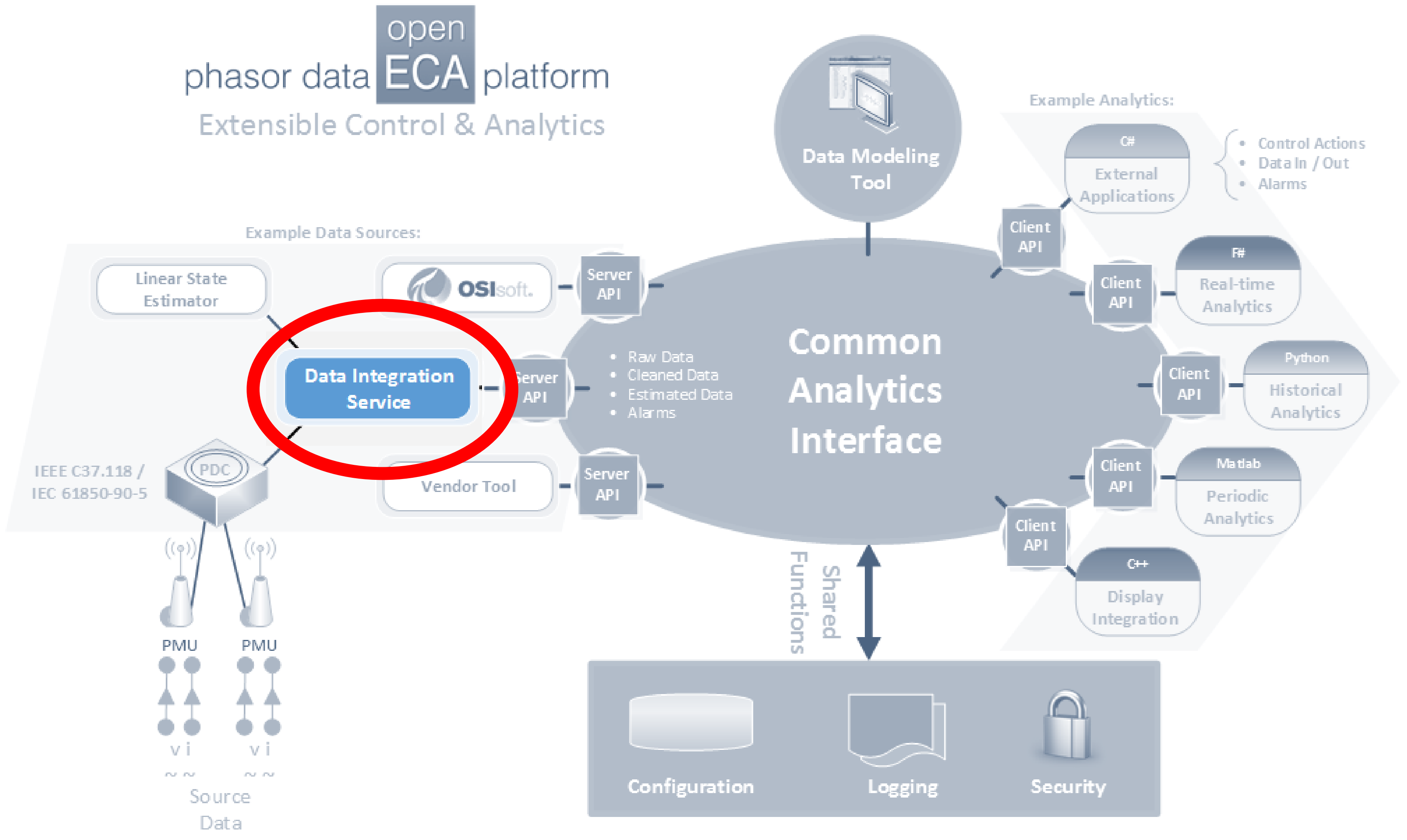
- Official project Schedule defines the following product releases:
  - Alpha – 12/31/2016
  - Beta – 4/30/2017
  - Version 1 – 9/30/2017

TEST HARNESS

openECA

DATA INTEGRATION SERVICES

Data Integration Services



Data Service Components

- Data Service
- Data Service Manager
- Data Service Console
- Performance Historian
- Data Historian
- RDBMS Configuration

Data Service

- Extensible adapter based architecture will allow integration of multiple data sources
  - Real-time (e.g., PMU, SCADA, State Estimation, Analytic Results)
  - Historical (e.g., OSI-PI Historian, local archives, SQL databases)
  - Semi-static (e.g., COMTRADE or PQDIF files)
- Maps data sources to time-series values
  - Support for common fundamental data types (e.g., float, int, bool, string)
- Common host for openECA Server API

Data Service Functions

- Measurement Definition
  - Flexible Data Types
  - Automatic Creation from Phasor Sources
- Measurement Validation
  - Data Quality Testing
  - Flat-line Detection
- Data Acquisition and Routing
- Adapter Configuration Management
- Linear State Estimation Value Integration
  - Available via secondary install and configuration
  - Allows grouping of actual and estimated values

Data Service Manager

- In the Test Harness prototype, the Data Service Manager is currently designed as a desktop client for configuration and monitoring of the Data Service – for future versions we intend on this being web based
- The Manager will be used to make connections to remote systems, e.g., Phasor Data Sources and SCADA systems as well as configure system parameters
- The Data Service will require an RDBMS system for storage of system configuration – the manager tool will be used to manage this configuration

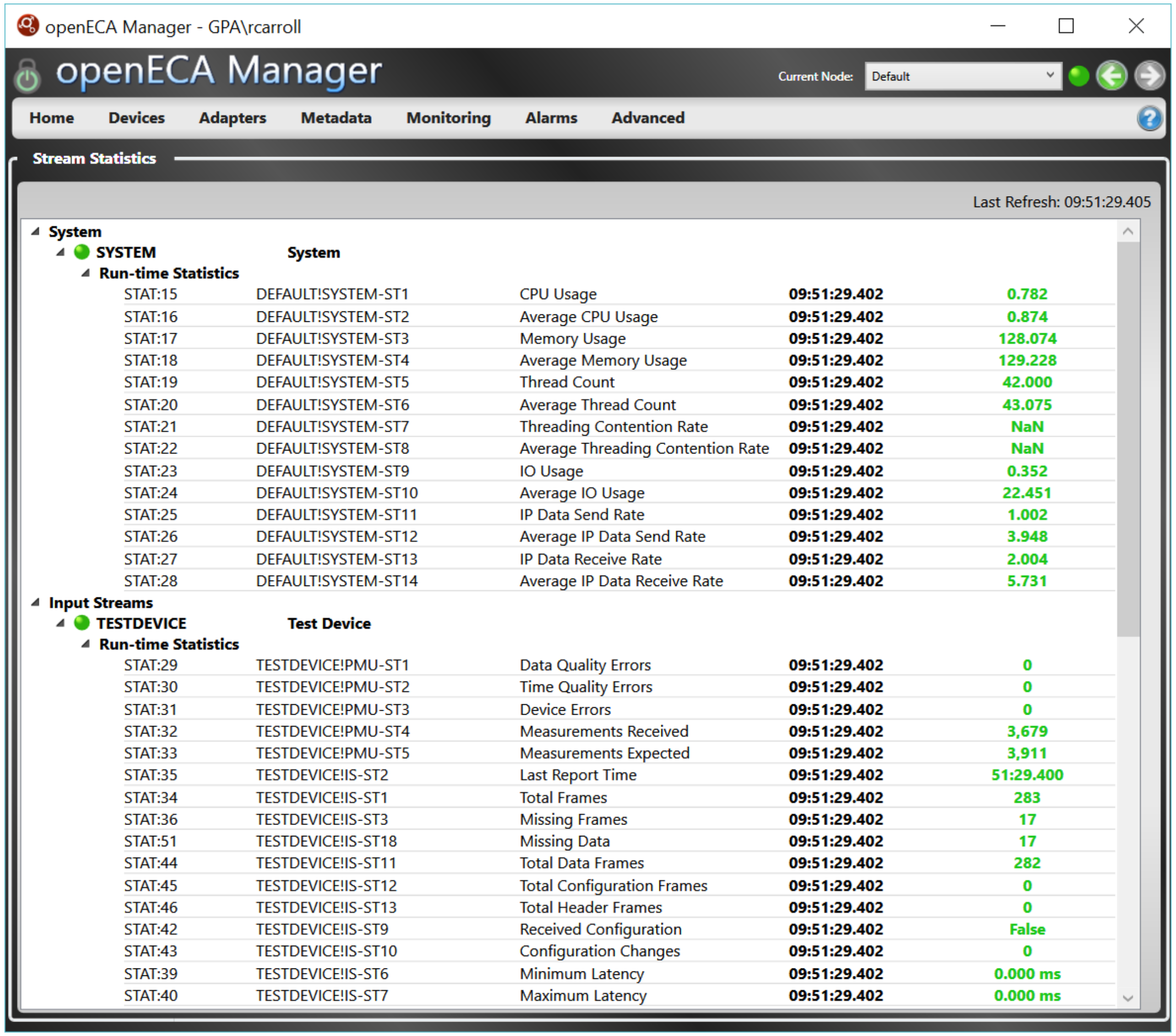
Data Service Console

- The Data Service Console application is a diagnostic tool used for monitoring the Data Service for current system activities as well as issuing low level system commands as may be needed during production operation
- The Console application will be typically focused towards IT professionals that will use the tool to quickly ascertain status of key system adapters

Performance Historian

- An local instance of the openHistorian will automatically deployed with the Data Service for archival of device and connection statistics for ongoing analysis, troubleshooting and auditing
- Multiple statistics will be gathered every ten seconds from all input, action and output sources

Performance Historian Example Metrics



Last Refresh: 09:51:29.405

System
- SYSTEM — System
  - Run-time Statistics

STAT:15	DEFAULT!SYSTEM-ST1	CPU Usage	09:51:29.402	0.782
STAT:16	DEFAULT!SYSTEM-ST2	Average CPU Usage	09:51:29.402	0.874
STAT:17	DEFAULT!SYSTEM-ST3	Memory Usage	09:51:29.402	128.074
STAT:18	DEFAULT!SYSTEM-ST4	Average Memory Usage	09:51:29.402	129.228
STAT:19	DEFAULT!SYSTEM-ST5	Thread Count	09:51:29.402	42.000
STAT:20	DEFAULT!SYSTEM-ST6	Average Thread Count	09:51:29.402	43.075
STAT:21	DEFAULT!SYSTEM-ST7	Threading Contention Rate	09:51:29.402	NaN
STAT:22	DEFAULT!SYSTEM-ST8	Average Threading Contention Rate	09:51:29.402	NaN
STAT:23	DEFAULT!SYSTEM-ST9	IO Usage	09:51:29.402	0.352
STAT:24	DEFAULT!SYSTEM-ST10	Average IO Usage	09:51:29.402	22.451
STAT:25	DEFAULT!SYSTEM-ST11	IP Data Send Rate	09:51:29.402	1.002
STAT:26	DEFAULT!SYSTEM-ST12	Average IP Data Send Rate	09:51:29.402	3.948
STAT:27	DEFAULT!SYSTEM-ST13	IP Data Receive Rate	09:51:29.402	2.004
STAT:28	DEFAULT!SYSTEM-ST14	Average IP Data Receive Rate	09:51:29.402	5.731

Input Streams
- TESTDEVICE — Test Device
  - Run-time Statistics

STAT:29	TESTDEVICE!PMU-ST1	Data Quality Errors	09:51:29.402	0
STAT:30	TESTDEVICE!PMU-ST2	Time Quality Errors	09:51:29.402	0
STAT:31	TESTDEVICE!PMU-ST3	Device Errors	09:51:29.402	0
STAT:32	TESTDEVICE!PMU-ST4	Measurements Received	09:51:29.402	3,679
STAT:33	TESTDEVICE!PMU-ST5	Measurements Expected	09:51:29.402	3,911
STAT:35	TESTDEVICE!IS-ST2	Last Report Time	09:51:29.402	51:29.400
STAT:34	TESTDEVICE!IS-ST1	Total Frames	09:51:29.402	283
STAT:36	TESTDEVICE!IS-ST3	Missing Frames	09:51:29.402	17
STAT:51	TESTDEVICE!IS-ST18	Missing Data	09:51:29.402	17
STAT:44	TESTDEVICE!IS-ST11	Total Data Frames	09:51:29.402	282
STAT:45	TESTDEVICE!IS-ST12	Total Configuration Frames	09:51:29.402	0
STAT:46	TESTDEVICE!IS-ST13	Total Header Frames	09:51:29.402	0
STAT:42	TESTDEVICE!IS-ST9	Received Configuration	09:51:29.402	False
STAT:43	TESTDEVICE!IS-ST10	Configuration Changes	09:51:29.402	0
STAT:39	TESTDEVICE!IS-ST6	Minimum Latency	09:51:29.402	0.000 ms
STAT:40	TESTDEVICE!IS-ST7	Maximum Latency	09:51:29.402	0.000 ms

Data Historian

- Users can integrate historian functionality, e.g., OSI-PI or the openHistorian, within the data service components.
- The integrated historian can archive data from incoming sources and also handle archival of analytic results
- Historical data can then be made available to analytics for processing archived events

RDBMS Configuration

- A relational database management system is required to store configuration information, e.g., measurement meta-data and device connection details, related to the Data Service
- The openECA Data Service supports the following common RDBM systems:
  - MS SQL Server
  - Oracle
  - MySQL
  - PostgreSQL
  - SQLite (no extra DB installation required)

TEST HARNESS

openECA

COMMON ANALYTICS INTERFACE

Common Analytics Interface Components

- Server API
- Client API
- Data Modeling Manager
  - Produces “Test Harness” Template

CAI Server API

- Configuration Serialization
  - Data class structure definitions
  - Labeling and identification of class instances with full measurement mapping
- Security Management
  - Validating clients and connections
  - Validating access to needed measurements
- Open API for Multivendor Support
  - Server API will define minimum requirements needed to implement a server side solution that will allow vendors to support analytics written using the openECA Client API

Target Platforms for CAI Server API

- Windows
  - Version 7 or greater
- Linux
  - Using Mono version 4 or greater
  - Will target latest Ubuntu for testing
- Mac OSX
  - Using Mono version 4 or greater

All options may not be available on all target platforms

Configuration Serialization

- Server will proxy all measurement meta-data provided by Data Service to Client API instances
- Server API will serialize identifiable data structures that are defined by the analytic developer using the Developer and Visualization tool – these data structures become the “inputs” and “outputs” of the analytics
    - User data structures will include mappings to measurements as defined in meta-data that is maintained by the Data Service

Security Management

- Using the security and authentication tools provided in the Shared Platform Services, the Sever API will authenticate connections from Client API instances
- The Server API will also validate access to data sets (i.e., groups of defined measurements) based on identity of authenticated user of Client API

CAI Client API

- User Defined Data Structure Definitions
  - Automatically referenced within analytic tools, the Client API will exist as a set of base services used to retrieve sets of user defined data for input and outputs
- User Defined Data Collection Windows
  - The Client API will handle time-alignment of incoming data and support creation of “windows” of data to an analytic, e.g., providing a one second window of data to an analytic every second

User Defined Data Structure Definitions

- Analytic developers will use the Development and Visualization tool to create custom collections of measurements that will form the data structures that will become the input and output of openECA
- A defined data structure will hold an instance of synchronized data all collected at a given time interval
- Custom data structures will contain:
  - Individual fields mapped to measurements, or
  - Arrays of fields with common identity, i.e., an array of measurements of the same type, e.g., all available synchrophasor frequencies

Short Term Analysis: Volatile Data Windows

- Although a user defined data structure only defines a collection of measurements for a single time instance, the CAI Client API will allow analytic developers to define windows of data, e.g., a full second of data, before calling the analytic function
- The CAI Client API will manage and call the analytic function with the data that has been collected on the user specified data window
- These collection windows will be stored in volatile memory and are a good option when the total data volume storage is reasonable for memory storage over the desired window

Long Term Analysis: Persisted Data Windows

- For longer data collection windows, the CAI Client API will persist collected data to disk, local to the machine running the analytic*
- Once the data collection window has expired, the analytic function will be triggered and allow the analytic function to read the collected data

* Internally this will just use a light-weight instance of the openHistorian and needed APIs for local persisted storage of structure data

Production Level Target Platforms

- The following platforms will be supported for analytic algorithm development and are considered applicable for production level deployments:
  - .NET (C# / F# / VB)
  - C++
  - Java

Test Level Target Platforms

- As time permits, the following platforms will be added for analytic algorithm development but should normally only be considered applicable for algorithm validation and testing:
  - MATLAB
  - Python
  - JavaScript
  - MS Excel

Integrating 3rd Party Analysis Tools

- Depending on target platform, it will often be useful to use existing libraries to speed analytic development – for more production platform deployments, projects templates will offer references to common handy third party tools, e.g.:
  - .NET Platforms:
    - Math.NET
    - Deedle
  - C++
    - Eigen

Data Modeling Manager

- Allows definition of logical groupings of measured values that represent organized structures of data, i.e., *data structures* – these definitions will map directly to language specific equivalents, e.g., a C language struct, an F# type or a class in C++, C# or Python
- Creates new uniquely labeled *identifiable instances* of data structures that directly map time-aligned measurement values to the structure fields – this will be auto-generated code, in the target language, that handles measurement-to-structure field mapping
- Using a real-time connection to the Server API, provides a *visual representation* of user defined data structures with updating values and simple trending with easy measurement lookup tools for mapping measurements to structure fields

Target Platforms for Data Modeling Manager

- Windows
  - Version 7 or greater
- Linux
  - Using Mono version 4 or greater
  - Will target latest Ubuntu for testing
- Mac OSX
  - Using Mono version 4 or greater

All options may not be available on all target platforms

TEST HARNESS

openECA

PREREQUISITES AND INSTALLATION

Tools needed to install and use the openECA prototype Data Modeling Manager and C# based analytic template, i.e., the "Test Harness" application to be used for algorithm development.

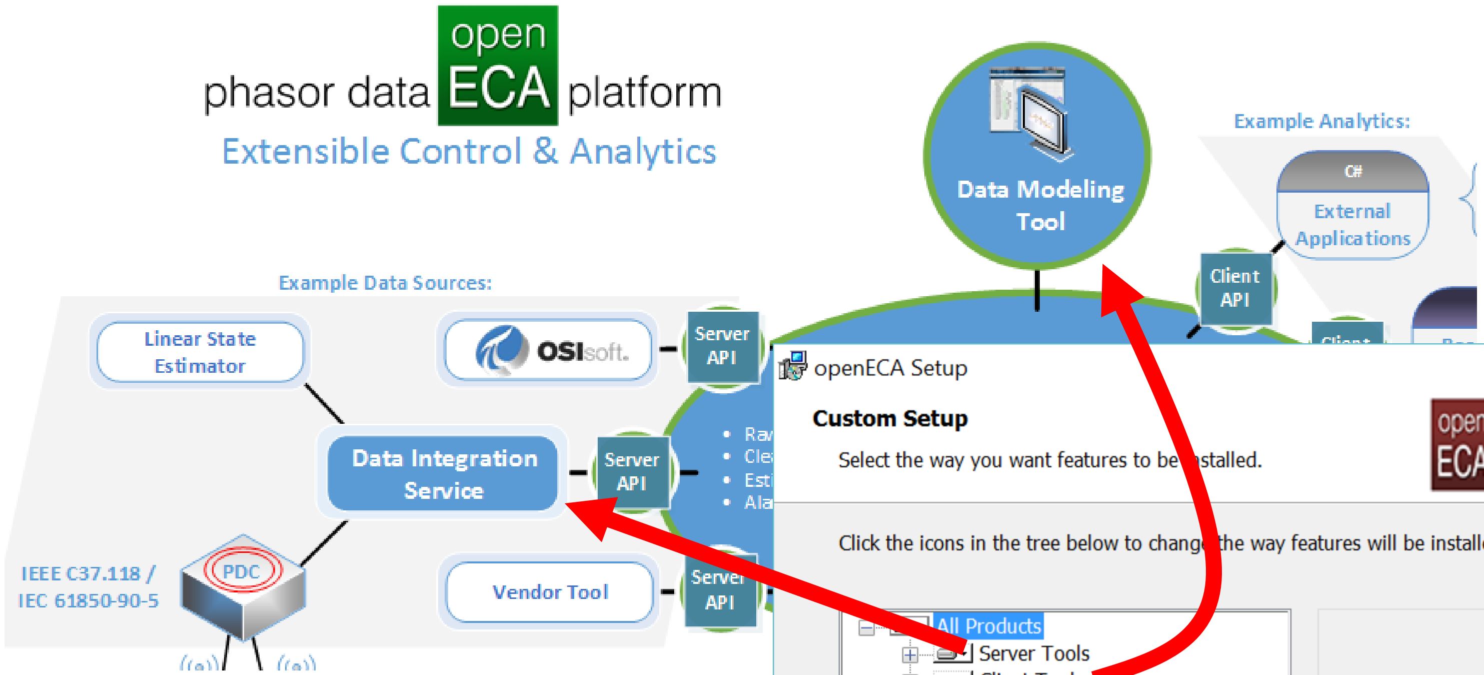
Development Environment

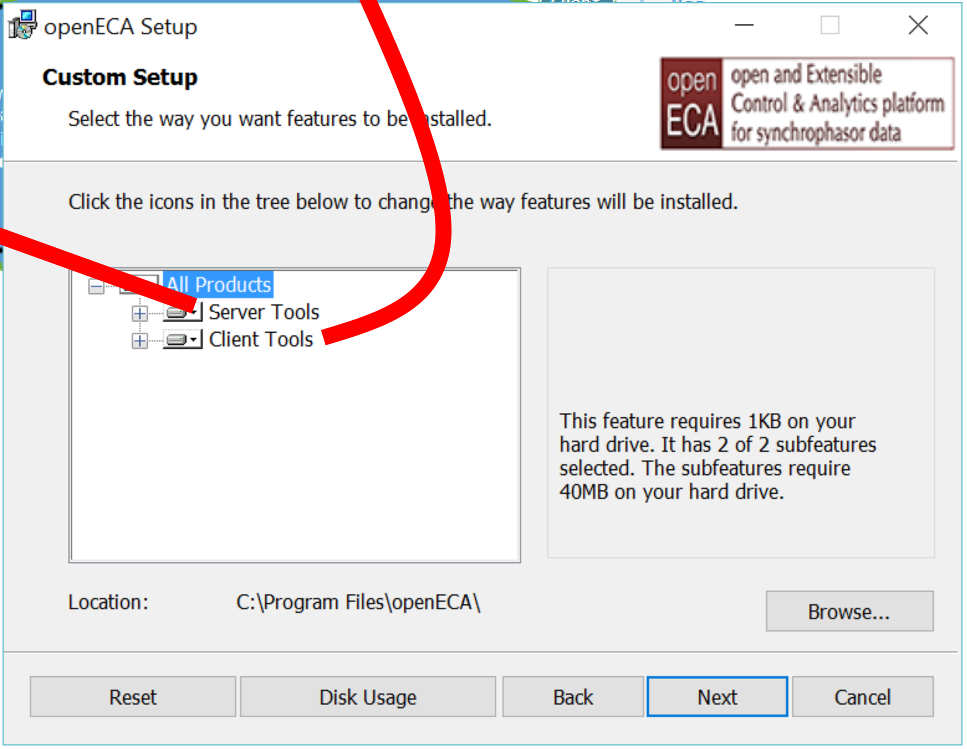
- For this prototype only C# is supported, so Visual Studio 2015 is recommended for primary analytic development and testing – the free Community Edition is fine
- Web based UI components target HTML 5 compatible browsers. The prototype has been developed using Google Chrome, so this browser is recommended, however, newer versions of Internet Explorer or Firefox may work fine, but these options have not been exhaustively tested

Installation Steps

- Download openECA Prototype with Analytic Development Test Harness from:
    - https://github.com/GridProtectionAlliance/openECA/releases
- Extract downloaded zip file contents into their own folder and run Setup.exe
- Note that Windows “Smart Screen” may show message that this application downloaded from the Internet is not trusted, you will need to select “Run Anyway”

Installation Options




openECA Setup

Custom Setup

Select the way you want features to be installed.

open ECA open and Extensible Control & Analytics platform for synchrophasor data

Click the icons in the tree below to change the way features will be installed.

- All Products
  - Server Tools
  - Client Tools

This feature requires 1KB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 40MB on your hard drive.

Location: C:\Program Files\openECA\

TEST
HARNESS

openECA

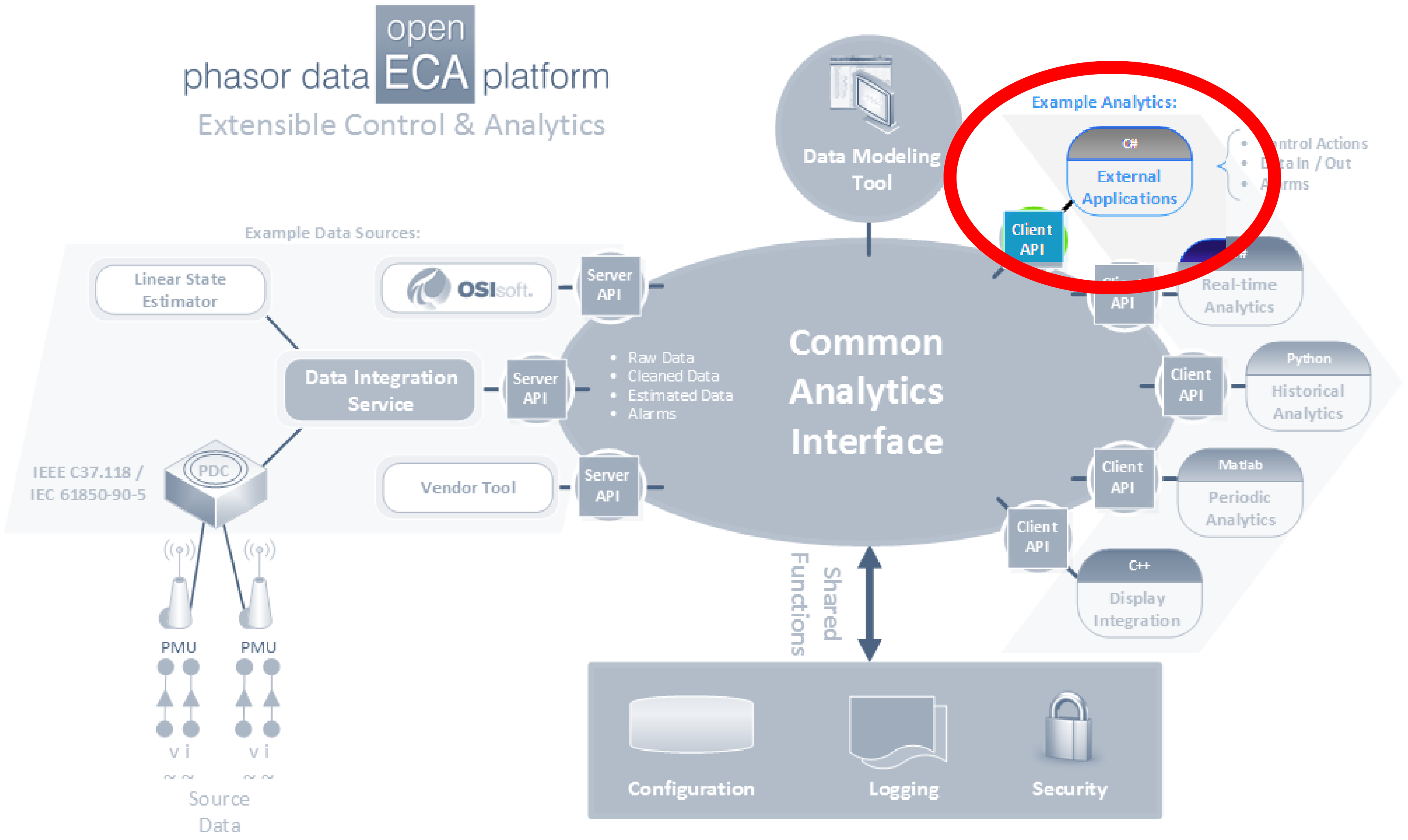
USE CASE

Creating a new analytic tool using openECA to provide phasor data

With openECA, a Paradigm Shift

- openECA defines a unified environment for modeling an analytic’s:
  - Configuration
  - Data Structures and
  - Measurement Mapping
- openECA will use a common user interface to deploy analytics templates to most modern development tools and be used to help develop and debug an analytic without adversely affecting existing production data collection environments

Analytic Development Test Harness



Creating a New Analytic

- Development of a new analytic will begin with the openECA openECA Data Modeling Manager
- The analytic developer will begin by using the tool to make a connection to one or more Server API instances and start reviewing the available data sources and associated measured values
- The manager will be used to create data structures and mappings to streaming data sources, then the developer will select a target language for their analytic and use the tool to create a new "Test Harness" project

Use Case Objective: Calculate Power

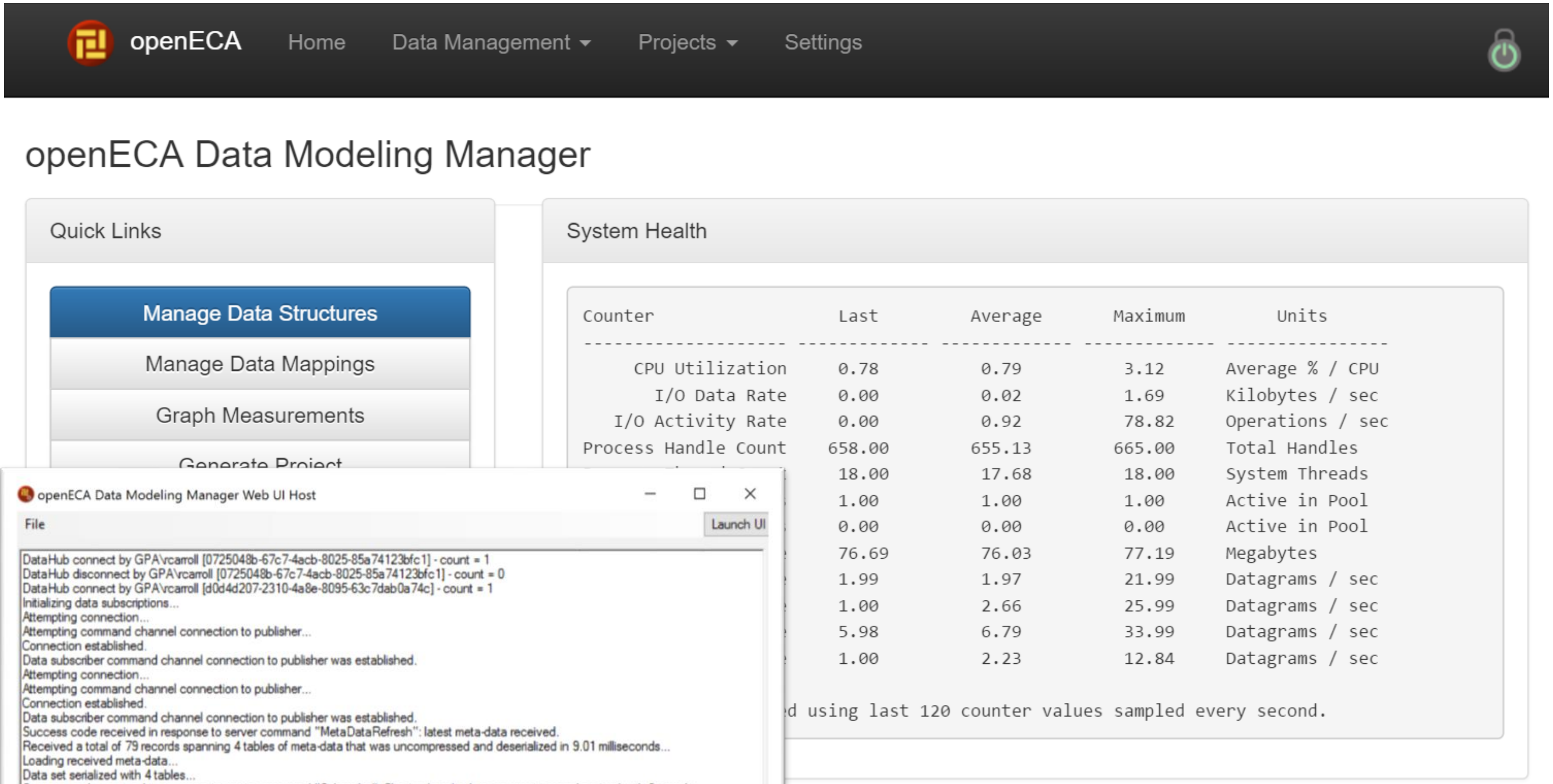
- We will walk through an example to calculate power, i.e., Active, Reactive and Apparent power, or, MW (P), MVar (Q) and MVA (S)
- For example, the equation to calculate MW is as follows:
  - 3 * voltageMagnitude * currentMagnitude * Math.Cos(voltageAngle - currentAngle) / 1.0e+6D;

Analytic Data Structure Definition

- The Data Modeling Manager allows the analytic developer to define classes of data structures, including the primary input and output:
  - We call the first domain input ***SourceData****
  - The second is the analytic product called ***ResultData****
- The contents of these data structures is under the complete control of the analytic developer
- The *SourceData* will be automatically populated with time-aligned incoming data over the desired interval, in this case, 1/30 of second
  - Auto-generated code that populates the *SourceData* structure is available for the analytic writer to review and validate
- The *ResultData* will be populated by the analytic developer and is returned at the end of processing (literally the return value from the primary function)

* Analytic developer will be able rename these data structures

Open the Data Modeling Manager Tool



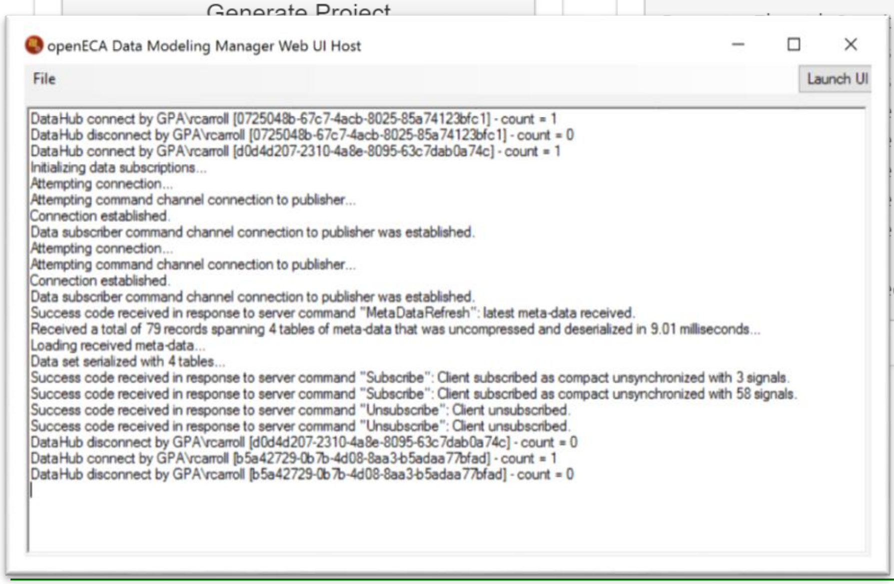
openECA Data Modeling Manager

Quick Links

System Health

```
Counter                  Last        Average      Maximum       Units
-------------------- ------------ ------------ ------------ ----------------
     CPU Utilization     0.78         0.79         3.12      Average % / CPU
       I/O Data Rate     0.00         0.02         1.69      Kilobytes / sec
   I/O Activity Rate     0.00         0.92        78.82      Operations / sec
Process Handle Count   658.00       655.13       665.00      Total Handles
                        18.00        17.68        18.00      System Threads
                         1.00         1.00         1.00      Active in Pool
                         0.00         0.00         0.00      Active in Pool
                        76.69        76.03        77.19      Megabytes
                         1.99         1.97        21.99      Datagrams / sec
                         1.00         2.66        25.99      Datagrams / sec
                         5.98         6.79        33.99      Datagrams / sec
                         1.00         2.23        12.84      Datagrams / sec

d using last 120 counter values sampled every second.
```



```
DataHub connect by GPA\rcarroll [0725048b-67c7-4acb-8025-85a74123bfc1] - count = 1
DataHub disconnect by GPA\rcarroll [0725048b-67c7-4acb-8025-85a74123bfc1] - count = 0
DataHub connect by GPA\rcarroll [d0d4d207-2310-4a8e-8095-63c7dab0a74c] - count = 1
Initializing data subscriptions...
Attempting connection...
Attempting command channel connection to publisher...
Connection established.
Data subscriber command channel connection to publisher was established.
Attempting connection...
Attempting command channel connection to publisher...
Connection established.
Data subscriber command channel connection to publisher was established.
Success code received in response to server command "MetaDataRefresh": latest meta-data received.
Received a total of 79 records spanning 4 tables of meta-data that was uncompressed and deserialized in 9.01 milliseconds...
Loading received meta-data...
Data set serialized with 4 tables...
Success code received in response to server command "Subscribe": Client subscribed as compact unsynchronized with 3 signals.
Success code received in response to server command "Subscribe": Client subscribed as compact unsynchronized with 58 signals.
Success code received in response to server command "Unsubscribe": Client unsubscribed.
Success code received in response to server command "Unsubscribe": Client unsubscribed.
DataHub disconnect by GPA\rcarroll [d0d4d207-2310-4a8e-8095-63c7dab0a74c] - count = 0
DataHub connect by GPA\rcarroll [b5a42729-0b7b-4d08-8aa3-b5adaa77bfad] - count = 1
DataHub disconnect by GPA\rcarroll [b5a42729-0b7b-4d08-8aa3-b5adaa77bfad] - count = 0
```

openECA Client app starts the web tool

Define Data Types for Power Calculator

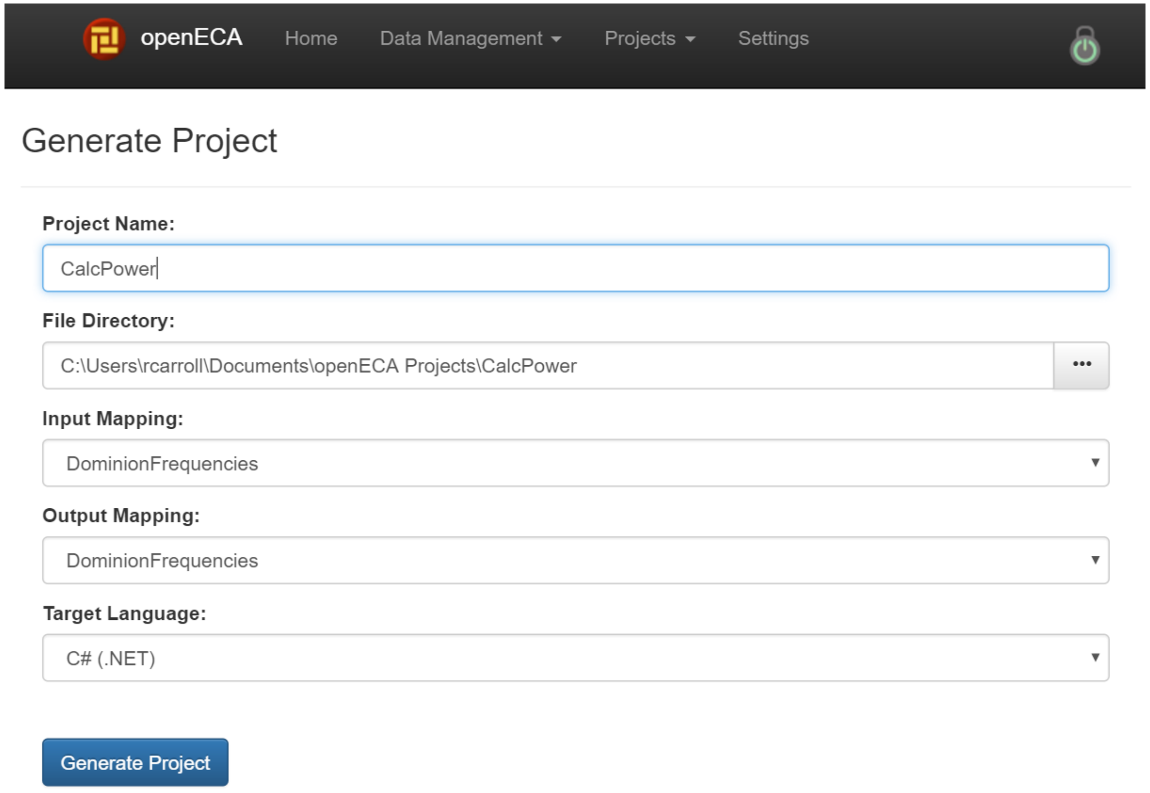
- Phasor (a complex number in polar form)
  - Angle (Double)
  - Magnitude (Double)
- VIPair (a voltage and current phasor)
  - Voltage (Phasor)
  - Current (Phasor)
- Power (MW/P, MVar/Q, and MVA/S)
  - Active (Double)
  - Reactive (Double)
  - Apparent (Double)

Define Mappings (Data Type “Instances”)

- TestDeviceV1 (Phasor)
  - Map Angle and Magnitude
- TestDeviceI1 (Phasor)
  - Map Angle and Magnitude
- SourceData (VIPair)
  - Map Voltage to TestDeviceV1, and
  - Map Current to TestDeviceI1
- ResultData (Power)
  - Map values to PPA:1, PPA:2, PPA:3

Generate Project

Click “Generate Project” and name application “CalcPower”



Generate Project

Project Name: CalcPower

File Directory: C:\Users\rcarroll\Documents\openECA Projects\CalcPower

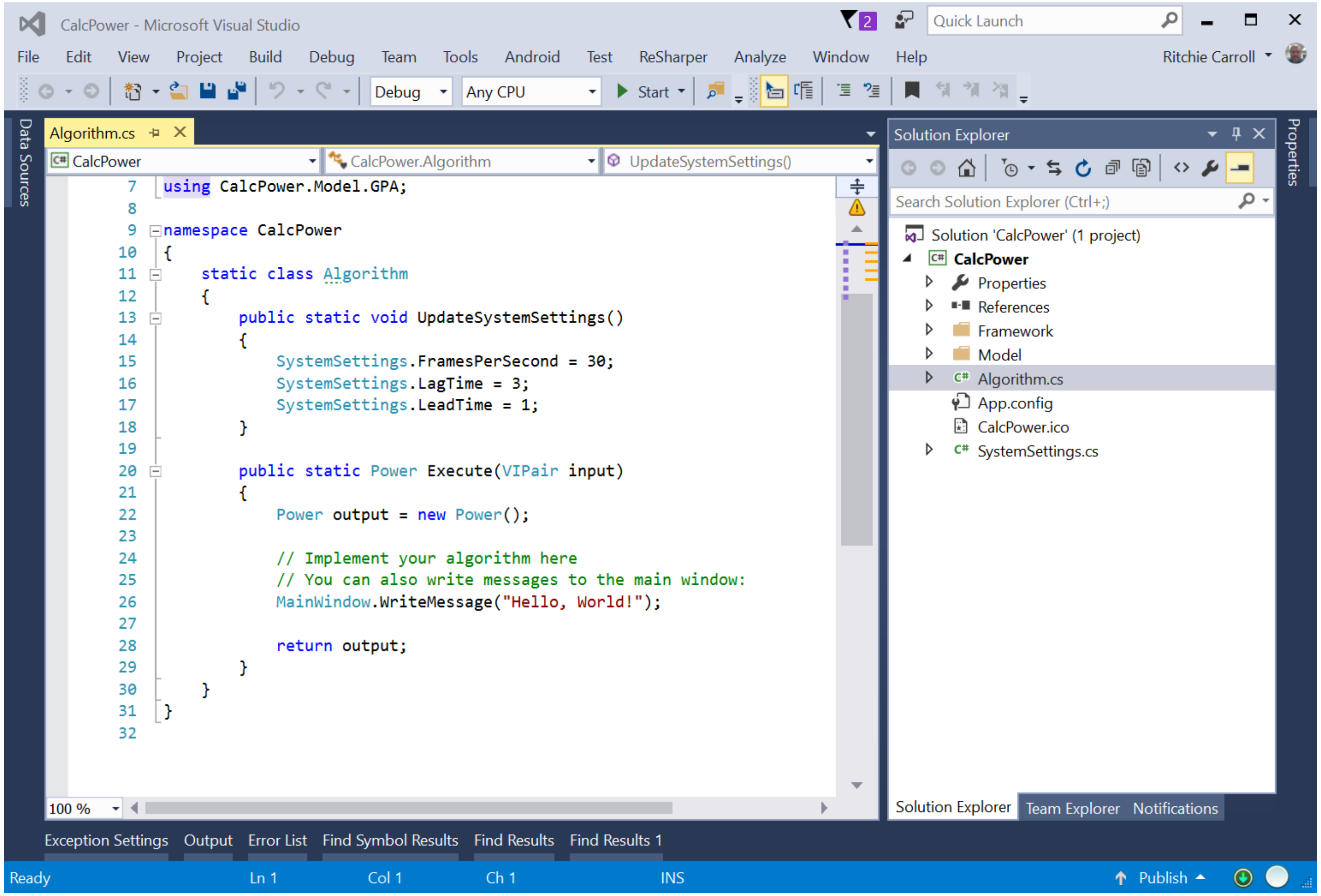
Input Mapping: DominionFrequencies

Output Mapping: DominionFrequencies

Target Language: C# (.NET)

Open Visual Studio Project

…\Documents\openECA Projects\CalcPower\



```csharp
using CalcPower.Model.GPA;

namespace CalcPower
{
    static class Algorithm
    {
        public static void UpdateSystemSettings()
        {
            SystemSettings.FramesPerSecond = 30;
            SystemSettings.LagTime = 3;
            SystemSettings.LeadTime = 1;
        }

        public static Power Execute(VIPair input)
        {
            Power output = new Power();

            // Implement your algorithm here
            // You can also write messages to the main window:
            MainWindow.WriteMessage("Hello, World!");

            return output;
        }
    }
}

```

Analytic Entry Point

Open “Algorithm.cs” and start adding code to single function:

ResultData Mapping

SourceData Mapping

```
Power Execute(VIPair input)
{
    // Analytic code goes here…
}
```

This function will be called every 1/30 of a second with a populated “input” structure.

Analytic Debugging

- Most modern development tools allow some level of native ability to debug code under development
- The openECA Client API will be providing data in the target language and be directly usable in the development tools
- Where supported by the development tool, this will allow the developer to put break points in their code, run the application, and stop on a line of code to evaluate the current data set and intermediate calculations while not adversely affecting the source systems

Analytic Deployment

- The *Test Harness* is a tool only for “developing” an analytic – the actual end product will be an installable Windows service or Linux Daemon
- Deployment of developed analytic will depend on language/platform used, however, the process usually involves copying compiled binary forms of the source code onto a host system, often using an installation script or application
- These deployments will include the analytic binaries along with Client API binaries for the target language
- Since it will be necessary to “map” analytic source data to different sources in different environments, deploying the openECA Analytic Modeling Tool along with the analytics will be important
- openECA will provide installation applications / scripts to make sure the analytic modeling tool can be easily deployed into target environments along with the analytic binaries

Visualization Development

- Visualization applications can also benefit from the openECA platform
- Development of visualizations would proceed identically to those steps laid out for analytic development
- One difference would be that a visualization does not usually provide results, so in this case the ResultData structure for a visualization would simply be an empty structure (often null or void depending on language)