Using Subscription API's

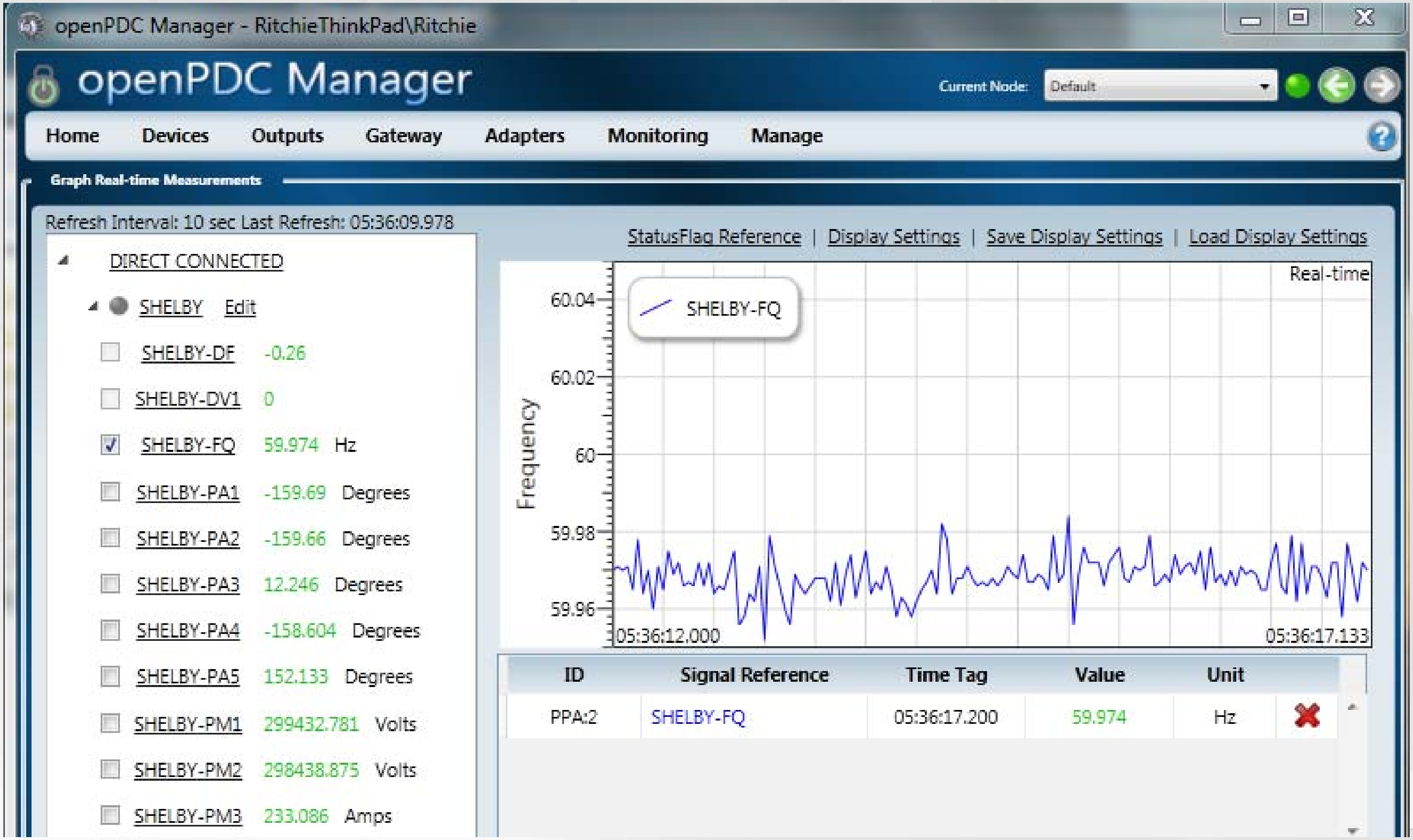
2012 User's Forum Tutorial

J. Ritchie Carroll / Stephen Wills

8/21/2012

GRID PROTECTION ALLIANCE

Subscribing to Measurements



Refresh Interval: 10 sec Last Refresh: 05:36:09.978

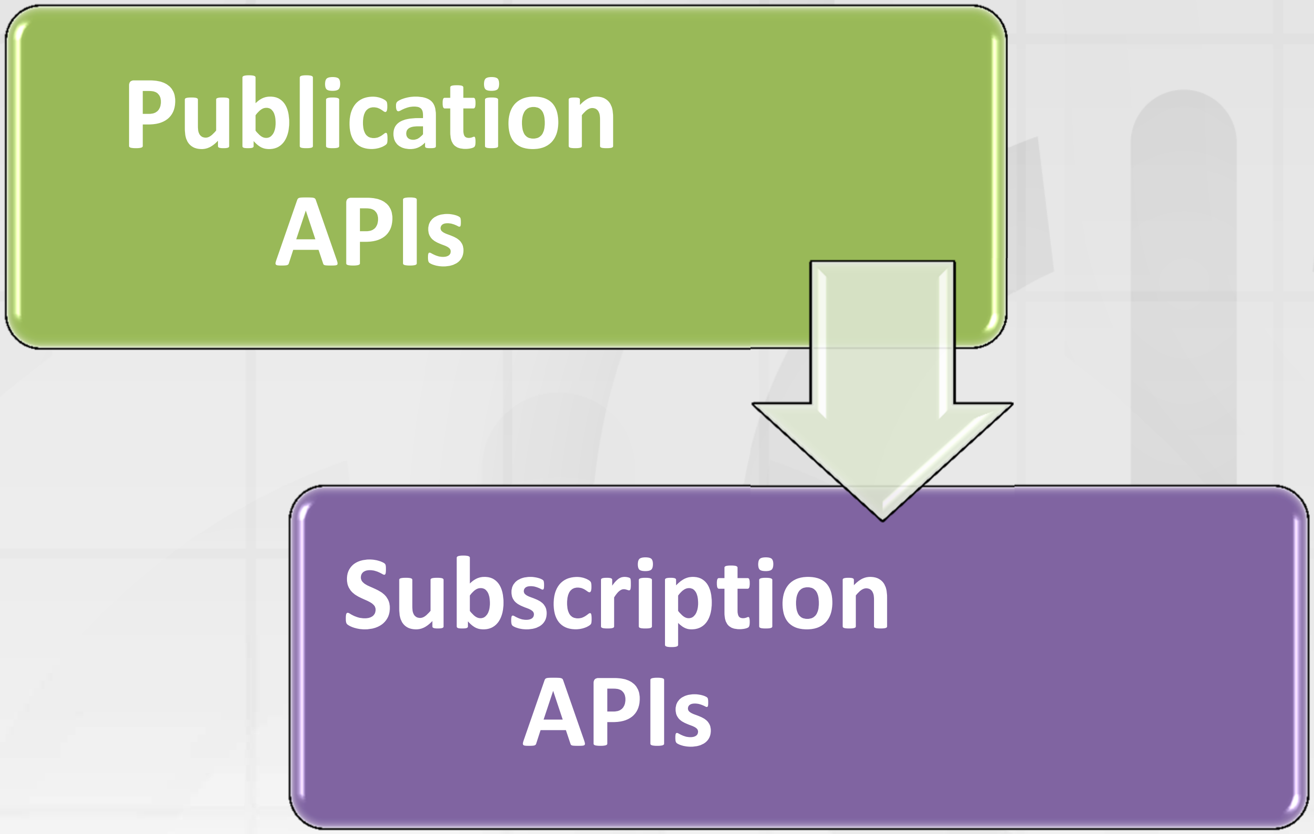
DIRECT CONNECTED

● SHELBY Edit

□	SHELBY-DF	-0.26	
□	SHELBY-DV1	0	
☑	SHELBY-FQ	59.974	Hz
□	SHELBY-PA1	-159.69	Degrees
□	SHELBY-PA2	-159.66	Degrees
□	SHELBY-PA3	12.246	Degrees
□	SHELBY-PA4	-158.604	Degrees
□	SHELBY-PA5	152.133	Degrees
□	SHELBY-PM1	299432.781	Volts
□	SHELBY-PM2	298438.875	Volts
□	SHELBY-PM3	233.086	Amps

ID	Signal Reference	Time Tag	Value	Unit	
PPA:2	SHELBY-FQ	05:36:17.200	59.974	Hz	✖

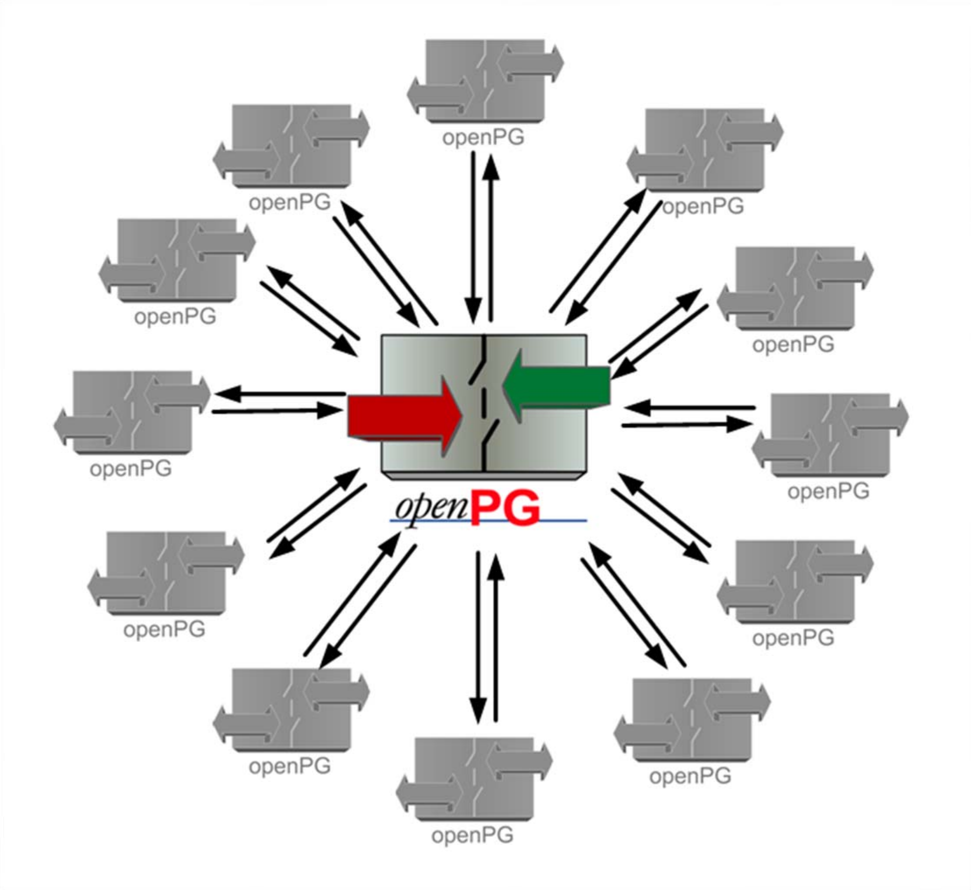
Primary Data Flow



Publication APIs

Subscription APIs

Performance Requirements



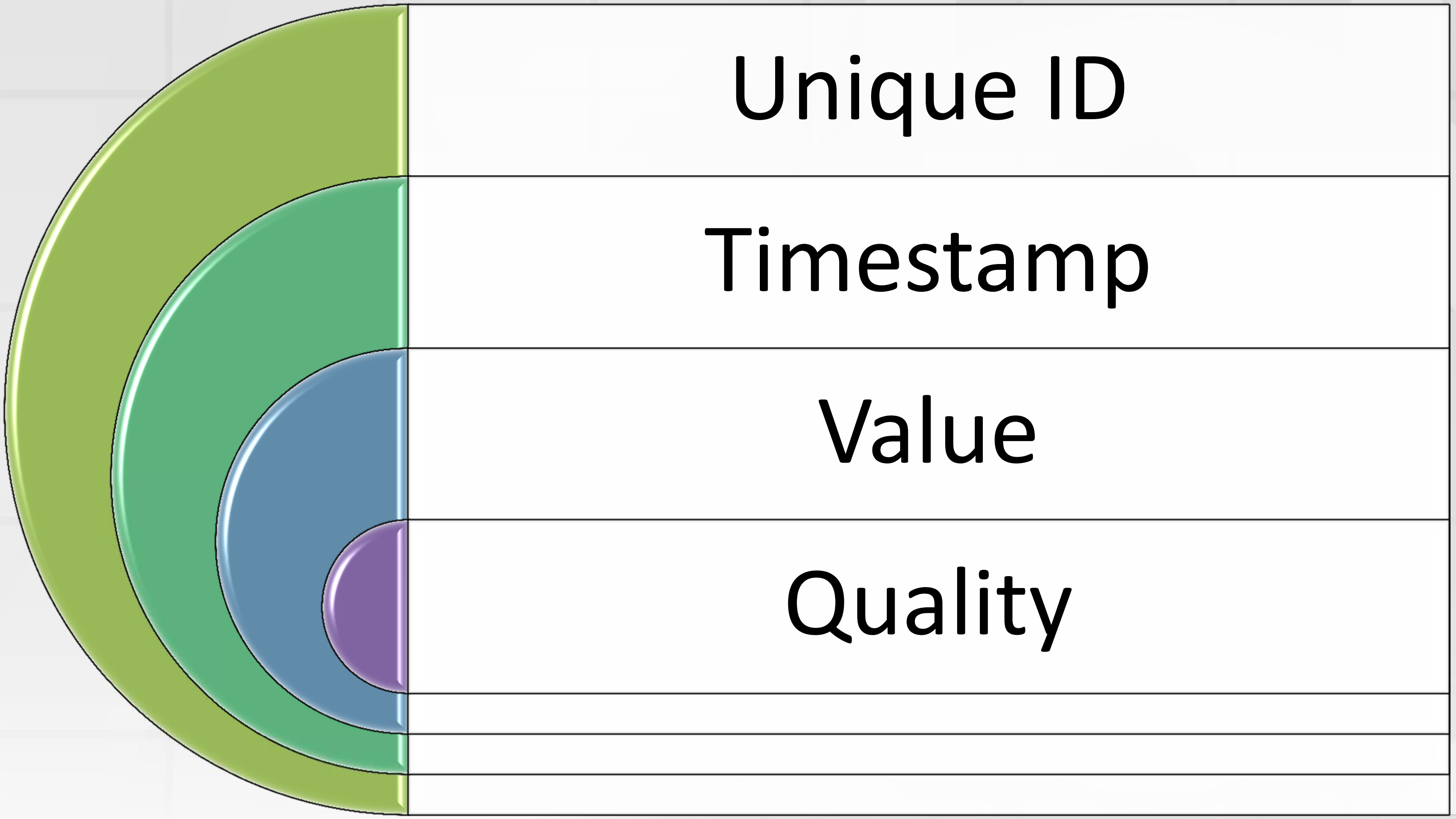
Publication and subscription APIs must move a *continually variable* set of points at low latency to be successful, around 1 million points per second.

Gateway Exchange Protocol (GEP)

- GEP is an extremely simple, small and fast wire format than can be used to exchange data points without a fixed predefined configuration – that is:
  - Points arriving in one data packet can be different than those arriving in another data packet. This can be due to each point having a different delivery schedule – or a dynamic schedule (e.g., alarms).
- GEP is a signal level publish/subscribe protocol with two available channels:
  - ***Command Channel (TCP)***
  - ***Data Channel (UDP or TCP)***

GRID PROTECTION ALLIANCE

Moving Measurement Data



Unique ID

Timestamp

Value

Quality

Simple Optimizable Structure

- Measurement data is well structured and can be safely condensed into a simple data structure (per signal):
  - ID
  - Time
  - Value (32-bit real number)
  - Flags
- There are many fast, highly effective lossless data compression opportunities for time-series data:
  - Simple 7-bit encoding can remove large volumes of “white space”
  - Due to the nature of the streaming measurment data, back-tracking compression methods can be highly effective

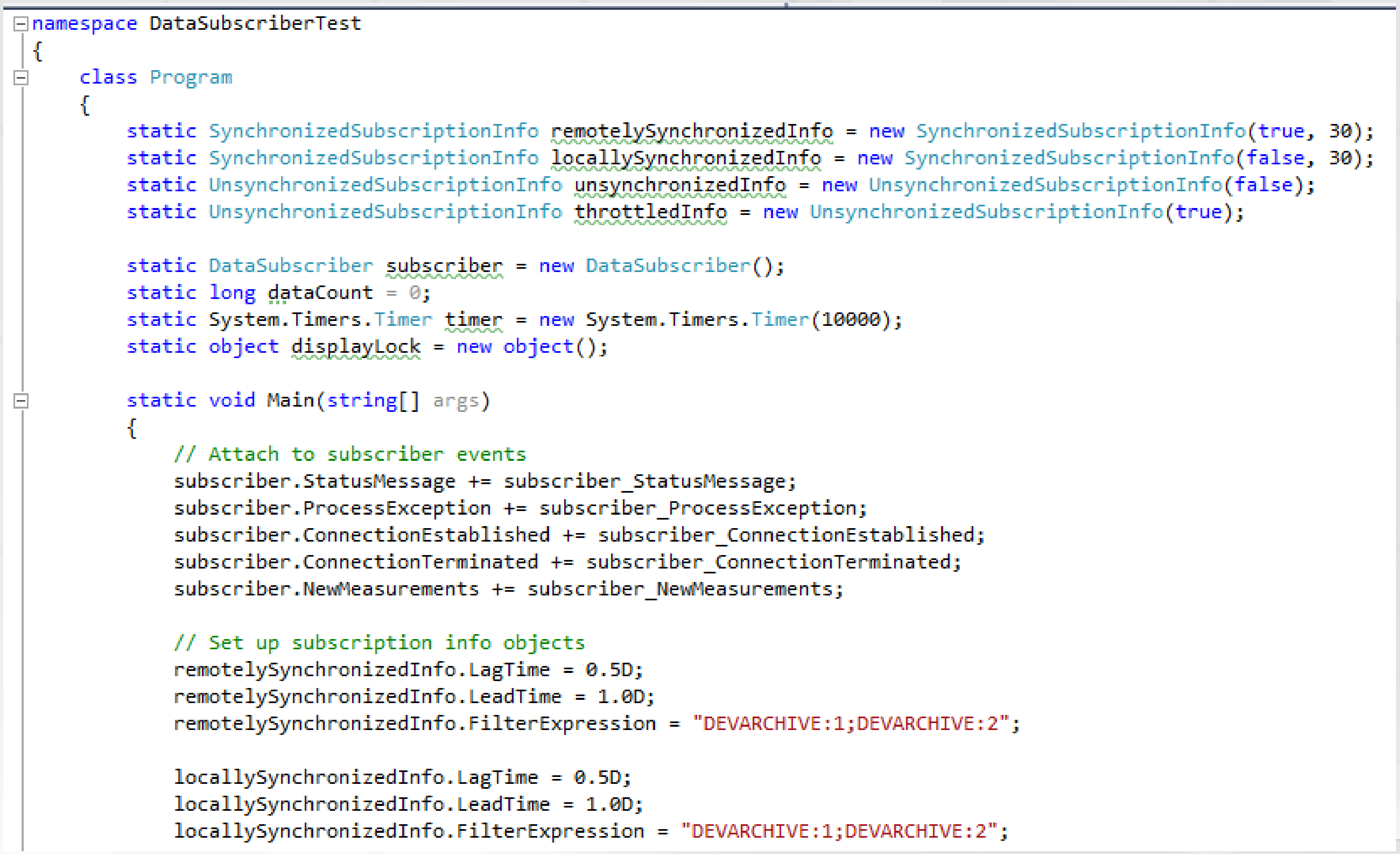
DataSubscriber API Usage

Purpose:

Receive

- **Attach to subscriber events**
- **Set up subscription info objects**
- **Initialize subscriber**
- **Start subscriber connection cycle**
- **Handle new measurement data**

Example DataSubscriber API Code



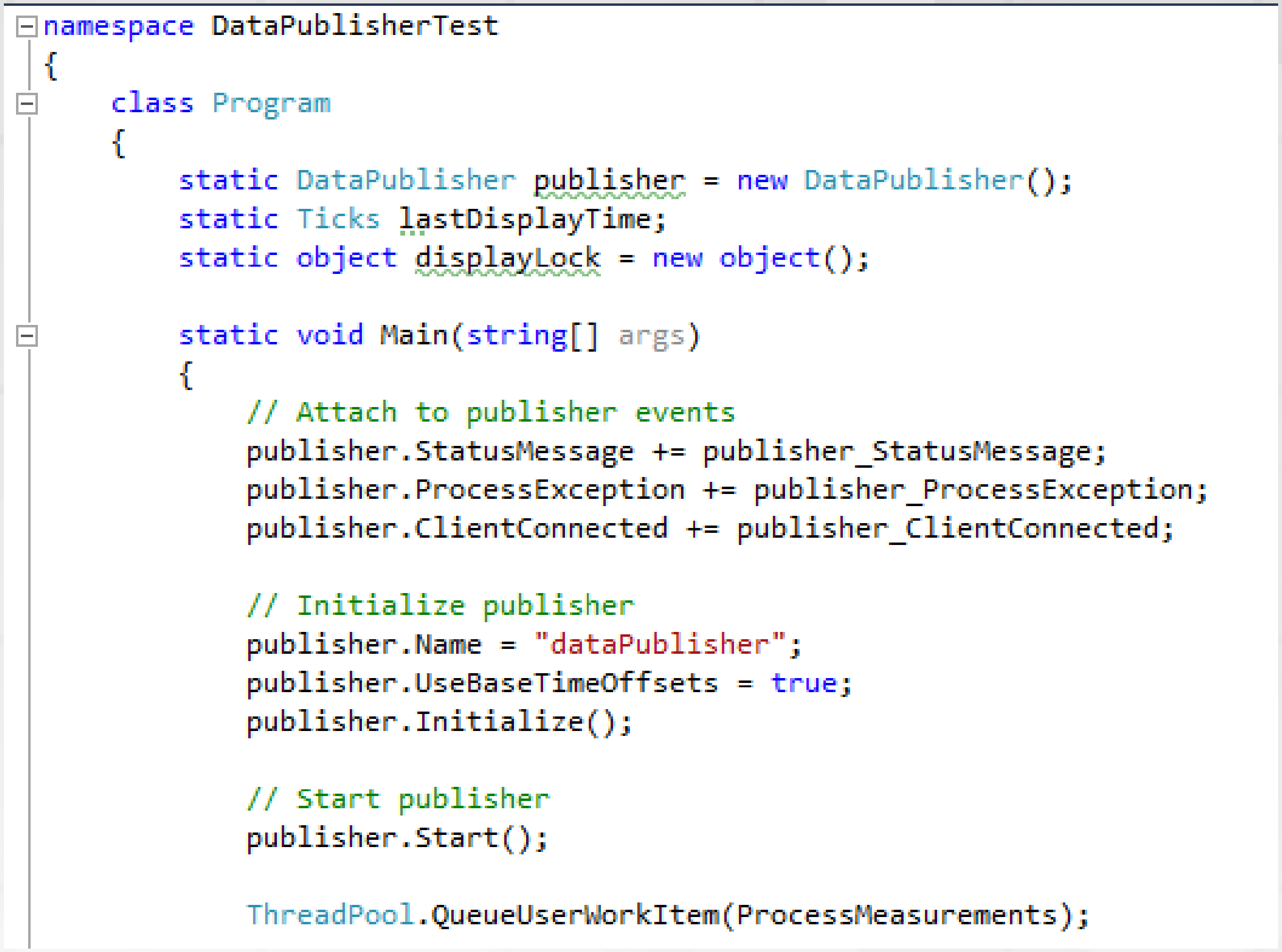
```
namespace DataSubscriberTest
{
    class Program
    {
        static SynchronizedSubscriptionInfo remotelySynchronizedInfo = new SynchronizedSubscriptionInfo(true, 30);
        static SynchronizedSubscriptionInfo locallySynchronizedInfo = new SynchronizedSubscriptionInfo(false, 30);
        static UnsynchronizedSubscriptionInfo unsynchronizedInfo = new UnsynchronizedSubscriptionInfo(false);
        static UnsynchronizedSubscriptionInfo throttledInfo = new UnsynchronizedSubscriptionInfo(true);

        static DataSubscriber subscriber = new DataSubscriber();
        static long dataCount = 0;
        static System.Timers.Timer timer = new System.Timers.Timer(10000);
        static object displayLock = new object();

        static void Main(string[] args)
        {
            // Attach to subscriber events
            subscriber.StatusMessage += subscriber_StatusMessage;
            subscriber.ProcessException += subscriber_ProcessException;
            subscriber.ConnectionEstablished += subscriber_ConnectionEstablished;
            subscriber.ConnectionTerminated += subscriber_ConnectionTerminated;
            subscriber.NewMeasurements += subscriber_NewMeasurements;

            // Set up subscription info objects
            remotelySynchronizedInfo.LagTime = 0.5D;
            remotelySynchronizedInfo.LeadTime = 1.0D;
            remotelySynchronizedInfo.FilterExpression = "DEVARCHIVE:1;DEVARCHIVE:2";

            locallySynchronizedInfo.LagTime = 0.5D;
            locallySynchronizedInfo.LeadTime = 1.0D;
            locallySynchronizedInfo.FilterExpression = "DEVARCHIVE:1;DEVARCHIVE:2";
```

DataPublisher API Usage

Purpose:
SEND

- **Attach to publisher events**
- **Initialize publisher**
- **Start publisher**
- **Queue new measurements for processing**

Example DataPublisher API Code



```
namespace DataPublisherTest
{
    class Program
    {
        static DataPublisher publisher = new DataPublisher();
        static Ticks lastDisplayTime;
        static object displayLock = new object();

        static void Main(string[] args)
        {
            // Attach to publisher events
            publisher.StatusMessage += publisher_StatusMessage;
            publisher.ProcessException += publisher_ProcessException;
            publisher.ClientConnected += publisher_ClientConnected;

            // Initialize publisher
            publisher.Name = "dataPublisher";
            publisher.UseBaseTimeOffsets = true;
            publisher.Initialize();

            // Start publisher
            publisher.Start();

            ThreadPool.QueueUserWorkItem(ProcessMeasurements);
```

Live Demos

- *Subscribing from a .NET C# application*
- *Subscribing from a C++ application (Linux)*
- *Subscribing from a Java application*