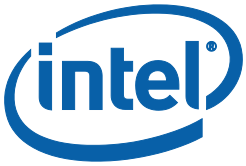


Intel® Metrics Framework

User's Guide

Document Number: 328139-001US

Contents

Legal Information

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to:

http://www.intel.com/design/literature.htm

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families: Go to: Learn About Intel® Processor Numbers

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

1. Introduction

Intel® Metrics Framework is a modular instrumentation framework that provides interfaces for providing and consuming metrics data. The API is implemented as a C++ library, which offers the following capabilities:

- You can add instrumentation sources to the framework independent of the data viewer.
- Instrumentation viewers can subscribe to metrics data independent of the data source.
- The framework implements a high-performance push model for delivering metric data to the consumers.
- The framework supports Windows* OS, OS X*, Linux* OS, and Android* OS environments.

This User Guide introduces the key concepts of the Intel Metrics Framework, enabling you to build an Intel Metrics Framework consumer and/or publisher extension that you can use to get various metrics data for further processing in your analysis tools or publish data, respectively.

For information on the Intel Metrics Framework API, refer to the "*Intel® Metrics Framework API Reference Guide*" available in the installation directory.

1.1 System Requirements

The Intel Metrics Framework supports the following software development environments:

Operating System	**Required Tools**
Windows 7*, Windows 8* (for both Desktop and Windows Store* applications); 32-bit/64-bit	Microsoft Visual Studio* 2010 Service Pack 1, or Microsoft Visual Studio 2012
OS X* 10.7, or higher; 32/64-bit fat binaries	Xcode* 4.6 or higher
Android* 4.0 or higher	Android* NDK, Revision 9
Ubuntu* 11.04 or higher; 32-bit/64-bit	Ubuntu* development packages 10.10 or higher

1.2 SDK Components

The Intel Metrics Framework provides the following key components:

SDK Component	**Description**
`gmframework.lib`	A static library that implements the key OS abstraction features, including a common dynamic extension loader with forward-backward compatibility support. All Intel Metrics Framework consumers, publishers, and client applications link to `gmframework.lib`. The SDK provides customized versions of this library for all of the supported operating system environments.
`igm_DefaultManager.dll` on Windows* OS `libigm_DefaultManager.so` on Linux* OS `libigm_DefaultManager.dylib`	This component is a dynamic extension module that manages Intel Metrics Framework extensions, such as publishers, conduits, and consumers. By providing this functionality in a dynamic extension, Intel Metrics Framework can support multiple versions of this component concurrently, to ensure backward compatibility.

on OS X*	
buildingblocks.lib	This static library provides commonly used code blocks that you can optionally use to build publisher, conduit, and consumer extensions more quickly.

All Intel Metrics Framework compatible applications using consumers, publishers, and/or conduits are based on these components and do not require any additional libraries.

1.3 Product Directories

To start using the Intel Metrics Framework, simply extract the archive to your system. The SDK directory structure is as follows:

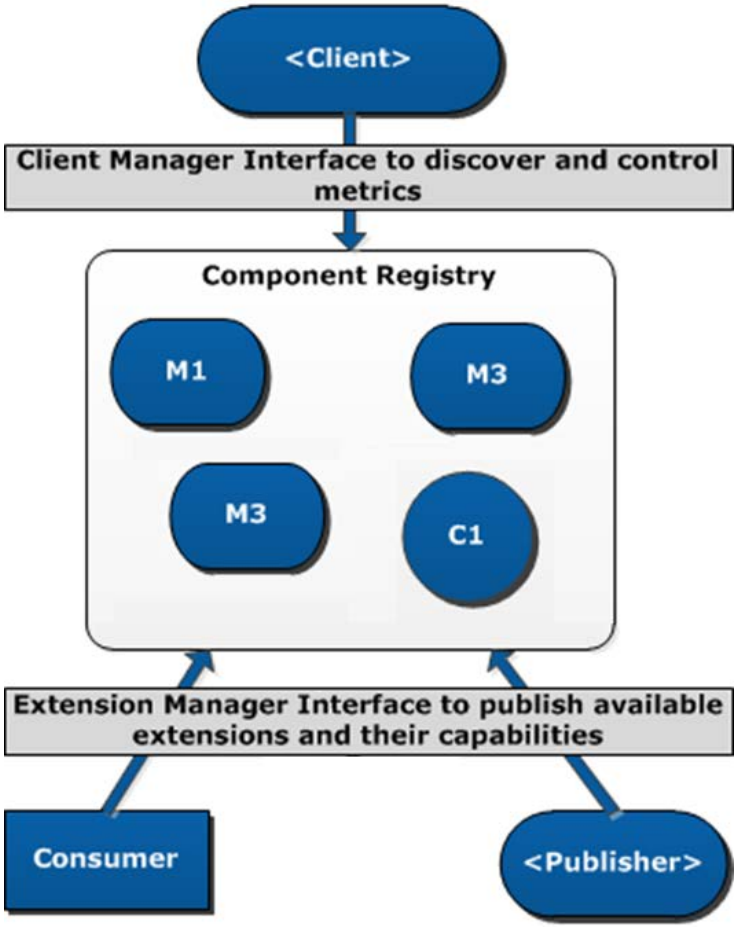
Directory	Description
bin	Contains the igm_DefaultManager dynamic-link library file in OS-specific subfolders.
doc	Contains the documents in this release.
include	Contains observation and building_blocks subdirectories holding the Intel Metrics Framework header files. When building custom extensions, you only need to include gmframework.h and building_blocks.h that reference all the other header files.
installers	Contains the redistributable installers for supported operating systems. The redistributable installers will install a set of publishers that provide various hardware metrics on Intel platforms. Please refer to Redistributable Publishers for more information on these publishers.
lib	Contains the gmframework and building_blocks static library files in OS-specific subfolders.
samples	Provides simple code examples that illustrate how to implement consumers and publishers using the Intel Metrics Framework.
tools	Contains built tools useful for Intel Metrics Framework development.

2. Architecture Overview

Intel® Metrics Framework provides a modular architecture that enables multiple extension libraries (consumers or publishers) and client applications to interact with the Intel Metrics Framework at run time.

An extension library extends the capabilities of the Intel Metrics Framework through the `IExtensionManager` interface. The Intel Metrics Framework, in turn, makes these capabilities available to client applications through the `IClientManager` interface. The component registry, which is accessible through the `IClientManager` interface, enables client applications to discover the capabilities offered by the extensions.

The following figure illustrates how different components of the Intel Metrics Framework work together:



where

- `M1`, `M2`, `M3` are metrics provided by the publisher. A client application can access these metrics as long as they are available in the component registry.
- `CU1` is a consumer object that enables the client application to receive metrics data.

Intel Metrics Framework supports the following usage models:

- You can extend the component registry to include component registries within other process and/or remote machines. See `IClientManager::ConnectRemote()` for details.

- Any client application can use the extensions from the component registry, regardless of where the consumers or publishers reside.
- Several client applications may use the same extensions simultaneously. However, an extension flagged as a singleton may only be loaded once on a given system.
- The component hierarchy can be nested. For example, a publisher can be a consumer of another metric stream at the same time.

The sections below provide definitions of the Intel Metrics Framework interfaces.

2.1 IExtensionManager

Extensions that implement publishers, conduits and/or consumers use the `IExtensionManager` interface to perform actions within the framework, such as publishing a list of available metrics to the component registry and pushing metric data through one or more conduits to the subscribed consumers.

2.2 IClientManager

Intel Metrics Framework uses this interface to control extensions and metrics subscriptions. The `IClientManager` interface enables you to:

- browse the component registry
- subscribe and unsubscribe consumers to/from metrics

2.3 IConsumer

The `IConsumer` objects represent a data end-point where inbound metric data is processed. A consumer object based on the `IConsumer` interface may be of two types:

- application-specific – provides metrics data in a customized format for client application UI
- general-purpose – writes data to a common file format, or computes an algorithm

An application subscribes one or more instances of `IConsumer` objects to desired metrics. While the subscription relationship persists, the consumer receives a notification whenever new metric data is available via the `OnMetricUpdated` callback function.

2.4 IPublisher

The `IPublisher` object describes the set of available metrics, including units, data payload types, compatibility information, detailed description and time basis. When a client application subscribes a consumer to metrics, the `IPublisher` object creates `IBoundMetric` objects. The `IPublisher` can dynamically add or remove a metric any time.

2.5 IBoundMetric

The `IBoundMetric` objects provides metric data to all consumers that request that data. Each `IBoundMetric` object is responsible for a specific metric.

3. Redistributable Publishers

The `installers/redist` directory in the SDK package contains installers for Windows* and Android* operating systems. Each installer will install a set of publishers that provide various metrics on Intel platforms. You can redistribute these publishers in your product. The recommended way to do this is to invoke the Intel® Metrics Framework installer within your product's installer. This will ensure that your product always provides the newest versions of the publisher binaries.

NOTE: The installer for Android* is an APK package that you can install onto your target device using the ADB* tool provided in the Android* SDK. After installing the Intel Metrics Framework APK, you must run the "Intel Metrics Framework" application once on the target device to complete installation.

The following table lists the publishers installed for each OS.

Name	Supported OS	Description
`DefaultManager`	Windows*, Android*	This is not a publisher, but is the runtime component that provides runtime services to the publishers and consumers.
`CPUInfoPublisher`	Windows	Provides various properties about the CPU on target platforms running on Intel processors, as attributes.
`GfxDrvSampledPublisher`	Windows, Android	Provides GPU metrics, in sampled mode, on platforms with Intel Processor Graphics microarchitecture codename Broadwell and later versions. Sampled mode includes the GPU usage for all processes running on the target system.
`GfxInProcPublisher`	Windows	Provides GPU metrics, in queried mode, on platforms with an Intel GPU, for applications that use Microsoft* Direct3D* 9, 10, and 11. In contrast with sampled mode, queried mode allows you to limit metrics collection to a specific application context, and you can control exactly the parts of the application's workload you want to measure. This allows you to analyze GPU usage for specific frames or draw calls within the application. But this requires that you load the publisher within the process space of the graphics application, and that you use a probe to denote the boundaries of the measurements at runtime. For more information on using this publisher, please contact Intel support at IntelPALSupport@intel.com.
`GPUInfoPublisher`	Windows	Provides various properties of the GPU hardware and driver on platforms with Intel® Processor Graphics, as attributes.
`MediaPerfPublisher`	Windows	Provides utilization metrics for the media processing parts of Intel GPUs, including both the fixed-function units and the programmable execution units.
`PowerPublisher`	Windows	Provides metrics related to power consumption on the target platform.
`RenderPerfPublisher`	Windows	Provides a frame rate metric for the overall system (desktop), individual Direct3D* applications, DXVA* 2.0 applications, and any Windows Store* applications

		running in fullscreen mode.
PVRPublisher	Windows	Provides GPU metrics on mobile platforms with a PowerVR* GPU.
SEPPublisher	Windows, Android	Provides access to a variety of raw hardware events (such as cache misses, branch mispredictions, or instructions retired) on Intel CPUs, enabling a low-level approach to platform monitoring. NOTE: The SEP Publisher documentation differs from the other publishers. See section 3.1.1 below for details. Also, the SEP Publisher provides configuration settings through environment variables – refer to section 3.1.2 below for details.
SampledGPUPublisher	Windows	Provides GPU metrics, in sampled mode, on platforms with the following Intel Processor Graphics versions: Intel® HD Graphics, Iris™, and Iris™ Pro graphics. Sampled mode includes the GPU usage for all processes running on the target system.

3.1 Obtaining Metrics Information

To obtain descriptive information about the metrics provided by the publishers listed above, use the doc_query option of the GM_Client tool. Since the list of metrics provided by the publishers depends on the platform, you should obtain the documentation on the target machine that you intend to analyze, after first running the redistributable installer on the target machine.

The following sections provide several usage examples of the doc_query option in GM_Client tool:

To display the name of each publisher:

```
GM_Client –r “doc_query publishers show name”
```

To display the name and description of each metric provided by all the publishers:

```
GM_Client –r “doc_query metrics show name_and_desc”
```

To display all the information for each metric provided by the PowerPublisher:

```
GM_Client –r “doc_query metrics from PowerPublisher show all”
```

For complete documentation on the doc_query option in GM_Client tool:

```
GM_Client –r “doc_query help”
```

3.1.1 Obtaining Information from SEP Publisher

The list of metrics supported by SEP Publisher is dynamic and depends on a target platform. You can use the doc_query option of the GM_Client tool (described above), to obtain the list of supported metrics.

Each of the SEP Publisher metrics is named in the following format:

```
com.intel.sep.<HW event name>[.core ID]
```

where,

- <HW event> is one of the hardware events
- [.core ID] portion of the name must be present for the events that are published per-core.

Examples:

Metric name: `com.intel.sep.INST_RETIRED.ANY.0`

Description: Assuming the default publisher settings, `com.intel.sep.INST_RETIRED.ANY.0` is a SEP Publisher metric that shows a number of samples for the core event `INST_RETIRED.ANY` that occurred on core `#0` during the single time interval.

Metric name: `com.intel.sep.UNC_CLOCK.SOCKET`

Description: Assuming the default publisher settings, `com.intel.sep. UNC_CLOCK.SOCKET` is a SEP Publisher metric that shows a number of times the uncore event `UNC_CLOCK.SOCKET` occurred during the single time interval.

For the detailed documentation of each hardware event, see:
Intel® 64 and IA-32 Architectures Software Developer Manuals

For some useful hardware events-based performance metrics, see:
Intel® VTune™ Amplifier XE documentation and white papers

3.1.2 SEP Publisher Configuration

Hardware events can be divided into two categories: core events and uncore events. Depending on a category that a particular event belongs to, there may be different ways that an event can be monitored. The following table summarizes the options for configuring the publisher:

Option	Core Events	Uncore Events	Environment variable	Default
Publish difference between previous and current value	Yes	Yes	`SEP_SEND_DIFF=1`	`SEP_SEND_DIFF=1`
Publish sampling data	Yes	No	`SEP_DO_EBC=0`	`SEP_DO_EBC=0` (if collected together, core events are published as samples, uncore events are always published as counters)
Publish event counters	Yes	Yes	`SEP_DO_EBC=1`	`SEP_DO_EBC=0`
Publish metrics per-core	Yes	Yes (not recommended)	Core events: `SEP_CORE_PER_CORE=1`, `SEP_UNCORE_PER_CORE=1`	`SEP_CORE_PER_CORE=1`, `SEP_UNCORE_PER_CORE=0`

- Core events can be either sampled or counted and published either per core or as a total number of samples or counts if publishing core metrics per-core is turned off (`SEP_CORE_PER_CORE=0`)
- Uncore events can only be counted. The uncore data arrives to `SEPPublisher` in series of counts. `SEPPublisher` selects the max count and publishes it – this is a current event count. It is possible to receive raw data by enabling the per-core publishing for uncore events (`SEP_UNCORE_PER_CORE=1`), however, this method requires proper handling and is difficult to understand.

4. Intel® Metrics Framework Tools

The tools described in this section are located in <SDK_root>/tools/<os>.

4.1 GM_Client

The GM_Client tool packaged with the Intel® Metrics Framework SDK can be used to show the component registry, to test metric data flow, and to provide metrics documentation (as described in the previous section).

The following sections provide several usage examples of the doc_query option in GM_Client tool:

To display the complete registry for all the publishers on the local machine:

```
GM_Client –c local –r registry
```

To display the complete registry after loading only MyPublisher on the local machine:

```
GM_Client –c local –f MyPublisher –r registry
```

To connect to the GM_Server running on the local machine at TCP port 4444, and display the data for the com.intel.gpu.gpu_gusy metric:

```
GM_Client –c socket –a localhost –p 4444 –m com.intel.gpu.gpu_busy
```

For complete command-line documentation on the GM_Client tool, run GM_Client on the command line with no parameters:

```
GM_Client
```

4.2 GM_Server

The GM_Server tool packaged with the Intel Metrics Framework SDK creates a server which will load all of the publishers that are available on that target system, based on the standard loading order. Your host consumer can then connect to that remote GM_Server instance and subscribe to the metrics published on that remote target system, just as you would subscribe to metrics published on the local host system.

CAUTION: Intel Metrics Framework server connections are not secured with cryptographic or secure handshaking mechanisms. Please ensure that your systems are protected with a firewall or other security mechanisms if this is a concern.

TIP: For added security, GM_Server provides a command line argument for IP address whitelisting. Only connections from specified IP addresses will be accepted.

```
>>GM_Server –p <port_number> -w (or –whitelist) <Path to a file containing a newline-separated list of IP addresses permitted to connect to the server>
```

Here are the steps to use GM_Server:

1. Copy the appropriate GM_Server and DefaultManager binaries for your platform to the device under test.
   NOTE: If running GM_Server on an Android* OS device, see the Android Debug Bridge (ADB) example below.

2. Copy any publisher libraries to the target system within the standard Intel Metrics Framework extension search path.
3. Run the GM_Server tool and specify either a socket port or a pipe name on the command line.

Once your consumer is connected to the conduit at the same socket or pipe, it can access all of the Intel Metrics Framework data on that target machine.

4.2.1 Executing GM_Server on an Android* OS Device

Running GM_Server from an Android* OS device requires the use of Google's Android* Debug Bridge (ADB) that is included in the Android* SDK. Here is a brief tutorial on using ADB to run GM_Server and transfer metric data back to a PC host:

1. Verify ADB is working and a device is present by executing the command "adb devices" from a command line on the host machine. This tutorial assumes you have exactly one device connected.
2. Create a working directory on the device by executing the command "adb shell mkdir /data/working_dir".
3. Copy the necessary binaries:
    a. "abd push GM_Server /data/working_dir"
    b. "abd push libigm_DefaultManager.so /data/working_dir"
    c. "adb push libigm_Waveform.so /data/working_dir" (Substitute for your own publisher. This example uses a Waveforms publisher)
4. Enable the execution bit for the GM_Server binary:
"adb shell chmod 777 /data/working_dir/GM_Server"
5. Forward an available port from your host machine to the device:
"adb forward tcp:6666 tcp:7777"
6. Execute the server: "adb shell /data/working_dir/GM_Server –port 7777"
7. Connect to the server using any Intel Metrics Framework client. For example, connect and subscribe to a metric using the GM_Client tool by executing the following command while the server is running: "GM_Client.exe -c socket -a localhost -p 7777 -m com.intel.samples.waveform_sin".

For details on ADB refer to: http://developer.android.com/tools/help/adb.html

4.2.2 Viewing Metrics in Intel® GPA System Analyzer

The Intel Metrics Framework includes the Intel® GPA System Analyzer tool that enables you to view metrics in graphical form. The GPA System Analyzer is available for Windows*, Ubuntu*, and OS X*, operating systems and can be found at <SDK_root>\tools\<os> in the SDK.

For instructions on installing System Analyzer on Ubuntu* OS, see http://software.intel.com/en-us/articles/intel-gpa-release-notes#ubuntu_install.

For instructions on installing System Analyzer on OS X*, see http://software.intel.com/en-us/articles/intel-gpa-release-notes#osx_install.

On Windows* OS, you do not need to install System Analyzer – you can simply run it in place.

To view metrics in System Analyzer:

1. Start an instance of GM_Server on your target system.
2. Start System Analyzer on your client machine.
3. If you are connecting to an Android* OS target system and a dialog box appears asking for the ADB path, enter it. If you are not connecting to an Android* target, click **Cancel**.
4. In the connection dialog box that appears, enter gm://<ip_address>:<port> using the IP address of the machine on which your GM_Server instance is running, and the port you specified when you started GM_Server.
5. Select **Connect** and then select **Local**.
6. Drag and drop metrics from the metrics list to the graphs.

5. Intel Metrics Framework Extensions

5.1 Code Samples

The following sample code shipped with the product can help you get started with the Intel® Metrics Framework. You can find these code samples in <*install-dir*>/samples/tutorial directory:

Sample	Description
TutorialOne	This sample enables you to: • create a simple publisher that provides a single metric • create a simple consumer that subscribes to the provided metric and prints the received values to stdout
TutorialTwo	This sample builds upon TutoralOne, and shows how to: • create several publishers that provide multiple metrics • create a consumer that enumerates all available metrics from any publisher • subscribe to metrics that the consumer can process and display to stdout
TutorialThree	This sample builds upon TutoralTwo, and shows how to: • create a server application that makes metrics from a single publisher available to one or more other applications through a TCP conduit • create a consumer that can subscribe to the provided metrics and display their values, in multiple concurrent processes

The next section guides you through TutorialOne, explaining how to create your own Intel Metrics Framework extensions.

5.2 Creating a Publisher

When you are ready to write your first publisher, you can use the Publisher project in the TutorialOne sample solution as a guide. To create a simple TutorialOnePublisher that provides a single metric, follow the steps in the sections below.

5.2.1 Edit the Publisher.h Header File

1. Include gmframework.h into your Publisher.h header file:

```
#include "gmframework.h"
```

   The gmframework.h header references all the headers required for both extensions and clients.

2. Make sure your publisher inherits the IPublisher interface and implements the IExtension::GetInstance() method. Typically, a publisher also needs to implement all the four IPublisher methods defined by the IPublisher interface:

```
class TutorialOnePublisher : public IPublisher
{
public:
```

```
        TutorialOnePublisher();
        virtual ~TutorialOnePublisher(){};
        // IExtension methods
        static IExtension * GetInstance();
        // IPublisher methods
        virtual void OnLoad();
        virtual void OnUnload();
        virtual IBoundMetric * CreateMetric(MetricHandle handle);
        virtual void DestroyMetric(MetricHandle handle, IBoundMetric * addr );
```

3. Add the `TutorialOneMetric` class that represents a "live" metric created by calling `IPublisher::CreateMetric()`. Make sure `TutorialOneMetric` inherits the `IBoundMetric` interface that enables binding the published metric to a conduit:

```
class TutorialOneMetric : public IBoundMetric
{
public:
   TutorialOneMetric();
   virtual ~TutorialOneMetric(){};

   // IBoundMetric members
   virtual void OnBound(ConduitHandle conduit, IWriteBuffer * buffer);
   virtual void OnUnbound(ConduitHandle conduit);
   virtual void OnFlush(ConduitHandle conduit){};
       // This publisher does not need to do anything on flushes
```

NOTE: Using the `OnFlush` method is optional. This method indicates that collected data should be delivered to the conduit. You can use this method when saving the data in an internal buffer instead of providing the data to the conduit when it is available.

5.2.2 Edit the Publisher.cpp File

1. Use the `EXPORT_EXTENSION` macro to expose an entry point in the extension binary object to the Intel Metrics Framework `framework`. The framework calls the `IExtension::GetInstance` entry point and uses Run-Time Type Information (RTTI) to determine which interfaces the object supports. In the code example below, `IExtension::GetInstance()` returns a new `TutorialOnePublisher` object, which supports the `IPublisher` interface:

```
BEGIN_EXTENSION_EXPORT
       EXPORT_EXTENSION(TutorialOnePublisher::GetInstance)
END_EXTENSION_EXPORT


IExtension * TutorialOnePublisher::GetInstance()
{
       return new TutorialOnePublisher();
}
```

2. Use the `IPublisher::OnLoad()` callback function as a notification that the framework has completed loading the extension binary. At this point, the extension can start interacting with the `IExtensionManager` interface:

```
void TutorialOnePublisher::OnLoad()
{
      IExtensionManager * mgr = GetExtensionManager();
      mgr->PublishMetric(this, MetricDesc("com.intel.tutorialone.sin",
      DM_OPCODE_TIME_STAMPED_DOUBLE), &m_HandleForMyMetric);
      DefineAttribute(this, "com.intel.tutorialone.sin\\Name", "Sin");
      DefineAttribute(this, "com.intel.tutorialone.sin\\Group",
"Tutorials");
      DefineAttribute(this, "com.intel.tutorialone.sin\\public", true);
}
```

In the code example above, the publisher uses the `OnLoad()` function as an opportunity to publish the supported metrics. Other publishers may publish metrics at a different time, or even dynamically change the list of metrics in response to a changing set of conditions. The `PublishMetric()` call takes a `MetricDesc` constructed from a URI and a type identifier for the metric data, and returns a handle for the metric `m_HandleForMyMetric`. This handle represents the published metric in the framework.

After publishing the metric, the `OnLoad` function defines a set of attributes associated with the metric.

NOTE: The attributes defined by the `OnLoad` function are not required by the framework. However, the `Name`, `Group`, and `public` attributes represent the set of attributes that all client applications can recognize. If you want your metrics to show up in Intel® Graphics Performance Analyzers, you must define all these attributes.

---

3. Use the `IPublisher::OnUnload()` callback function as a notification that the framework is ready to start unloading the extension binary. At this point, the publisher can clean up its resources. When the framework unloads a publisher it also unpublishes all the metrics of this publisher.

```
void TutorialOnePublisher::OnUnload()
{
   IExtensionManager * mgr = GetExtensionManager();
   mgr->UnpublishMetric( this, m_HandleForMyMetric );
}
```

4. Add a call to `IPublisher::CreateMetric()` to get a run-time instance of a published metric. Generally, this operation occurs when a client application binds a metric to a conduit. In the example below, the publisher returns a new instance of `TutorialOneMetric`, which implements `IBoundMetric`. The `IBoundMetric` interface represents the run-time instance of the created metric.

```
IBoundMetric * TutorialOnePublisher::CreateMetric(MetricHandle handle)
{
   return new TutorialOneMetric();
}
```

5. Add a call to `IPublisher::DestroyMetric()` for the framework to remove a given run-time metric instance that is no longer needed.

NOTE: The lifetime of a metric instance is determined by the publisher.

```
void TutorialOnePublisher::DestroyMetric(MetricHandle, IBoundMetric * addr)
{
   delete addr;
}
```

6. Add a call to `IBoundMetric::OnBound()` after the framework binds the metric to a conduit, in response to a binding request from a client application. The framework passes the conduit handle and the buffer in the `OnBound()` call. It is up to the publisher to keep track of the buffer that is used with each metric-to-conduit binding. When the framework makes a buffer available for data, the publisher should start sending metric data into the buffer. In the example below, this is the purpose of the thread that calls the `GenerationThread()` function. Use the `AutoLock` to serialize access to the `m_buffer_map`, since the map is updated by the thread that calls `OnBound()` and `OnUnbound()` and the map is referenced by `GenerationThread()`.

```
void TutorialOneMetric::OnBound(ConduitHandle conduit, IWriteBuffer * buffer)
{
   AutoLock lock(m_mapLock);

   if( m_buffer_map.empty() )
   {
         m_RunThread = true;
         m_thread = GMCreateThread((GMThreadFunc)GenerationThread, this);
   }

   m_buffer_map[conduit] = buffer;
};
```

7. Add a call to `IBoundMetric::OnUnbound()` before the framework removes the binding of the metric to a conduit. The metric delivery thread stops when no more buffers are available to receive the data.

```
void TutorialOneMetric::OnUnbound(ConduitHandle conduit)
{
    AutoLock lock(m_mapLock);
    m_buffer_map.erase(conduit);

    if( m_buffer_map.empty() )
    {
        m_RunThread = false;
        GMJoinThread( m_thread );
    }
};
```

8. Send metric data to each active buffer that the framework initialized using the `OnBound()` function. Use the `Encode_TIME_STAMPED_DOUBLE_IMPLICIT` function to encode the data: the `'TIME_STAMPED_DOUBLE'` part of the function name matches the `DM_OPCODE_TIME_STAMPED_DOUBLE` type identifier specified when you published your metric. The `'IMPLICIT'` part means that the system generates the timestamp value at the time of encoding.

```
{
    AutoLock lock(me->m_mapLock);
    for( BufferMap::iterator i = me->m_buffer_map.begin();
         i != me->m_buffer_map.end(); i++ )
        Encode_TIME_STAMPED_DOUBLE_IMPLICIT((*i).second, 1 + sin(me-
>m_Value));
}
```

5.2.3 Creating a Metric URI

All metrics must conform to the following Intel Metrics Framework URI naming standards. This ensures that metric name do not collide, and it should be easier for other users of your publisher to understand:

`com.intel.<metric_origin>.<metric_name>`

The GPA team currently uses and recommends the following values for `metric_origin`:

- `gpu`
- `cpu`
- `uncore`
- `os`
- `gm`
- `samples`

Examples:

- `com.intel.gpu.eu_active_in_vs`
- `com.intel.os.system_frame_rate`
- `com.intel.uncore.display_bandwidth`

5.2.4 Testing Your Publisher

To test your publisher, please refer to GM_Server for information on how to use the GM_Server tool. If you need to test a publisher on a rooted Android* OS device, see Executing GM_Server on an Android* Device.

6. Working with Multiple Product Versions

6.1 Versioning and Binary Compatibility

The Intel® Metrics Framework supports the following goals through effective versioning of SDK releases, files, and interfaces:

- Enable users and installation software to avoid inadvertently downgrading an existing Intel Metrics Framework installation.
- Support forward compatibility among binary components.

6.1.1 SDK Versioning

To uniquely identify different releases, the Intel Metrics Framework adopts the following versioning format:

<major>.<minor>.<build_number>

6.1.2 File Versioning

File versioning is currently supported only on Windows* and Android* operating systems.

Windows* OS

All Intel Metrics Framework executable files built for Windows OS (.exe files and .dll files, including the DefaultManager and the publishers) are stamped with a file version that matches the SDK version. File versioning information is specified in the `VS_FIXEDFILEINFO` resource structure embedded within each file. You can get this information using the `GetFileVersionInfo()` Windows* OS API.

Android* OS

The executable files built for Android* OS are not version-stamped. However, the Intel Metrics Framework SDK package includes an APK package with versioning information that matches the SDK version. You can use this package to install the Intel Metrics Framework components onto an Android-based target system. When you install the APK, the Intel Metrics Framework appears as an application on the device, with the associated version information.

6.1.3 Interface Versioning

The Intel Metrics Framework provides full backward compatibility among different releases at the Application Binary Interface (ABI) level. Once a programming interface such as `IConsumer` or `IPublisher` is introduced in the framework, that interface will remain unchanged in subsequent releases. When the Intel Metrics Framework introduces new functionality in later releases, it will always do so through a new interface. For example, if a future release extends the existing Publisher functionality, the new release will introduce the functionality with a new interface with a new name, for example, `IPublisher2`, while still supporting the existing `IPublisher` interface. When the Intel Metrics Framework introduces a new category of functionality, it will expose the functionality with an entirely new interface.

6.2 Installing Runtime Binary Files

To install the Intel Metrics Framework runtime binaries, that is, `DefaultManager` and the publishers, do the following:

- For a single application, install the extension binaries to your application's working directory.
- For multiple applications, install the extension binaries into a well-known folder that all your applications can access. To avoid degrading other installed applications that are already using the Intel Metrics Framework, make sure that you do not replace the extension binaries in the well-known folder with older versions. You can safely install newer versions since the Intel Metrics Framework is backward-compatible.
    - On Windows* OS, use the file versioning information to determine which files are newer. The `redist` installer (located in `\installers\redist\windows` in the SDK package) automates this process.
    - On Android* OS, this process is automated when you install the Intel Metrics Framework APK (located in `/installers/redist/android` in the SDK package). After installing the APK, you need to run the "Intel Metrics Framework" application once on the target device to complete installation.

6.3 Loading Extension Binary Files

When using multiple versions of the Intel Metrics Framework on the same system, make sure you are loading the correct extension binaries. By default, the Intel Metrics Framework `ExtensionLoader` looks for extensions in the following order, until it finds the requested extension:

- Directories added via the `ExtensionLoader::AddSearchPath()` function.
- The paths specified in the `GM_COMPONENTS` environment variable.
- The current working directory of the process calling the `ExtensionLoader`.
- On Windows* OS, the install location used by the Intel Metrics Framework redist installer (`C:\Program Files\Intel\Metrics Framework\<arch>`).

7. Using Intel Metrics Framework with Shared System Resources

In some situations, your publisher might need to access one or more shared system resources to collect metrics data. For example, the Intel® Processor Graphics driver can provide system-wide metrics in time-based sampling mode delivering metric data through an I/O stream that only a single application can read at a time. In this case, only one such publisher, within a single process, can publish such metrics and this publisher should be flagged as a singleton using the `EXPORT_SINGLETON_EXTENSION` macro. Multiple consumers can still access this data, if the publishing process makes the metrics available system-wide through a `SocketServerConduit`. The `TutorialThree` sample demonstrates this mechanism, using the `SocketConduitFactory` class provided in the Intel Metrics Framework `building_blocks` library.

NOTE: Singleton extensions will not be loaded within WinRT based applications due to the restrictions and sandboxing placed on such applications by the operating system.