The latest version of the tutorial is maintained online at:
 http://research.microsoft.com/en-us/projects/senseweb/SenseWebTutorial.pdf
Sample code is shared at:
 http://research.microsoft.com/en-us/projects/senseweb/Samples.zip
Last updated on June 18, 2009.

SenseWeb Tutorial

Lesson 1 - An Introduction to SenseWeb

This lesson includes introduction of basic SenseWeb concepts as well as overview of system architecture to help you get an idea of what SenseWeb is and how you could possibly use SenseWeb.

1.1 Introduction

SenseWeb, developed by Microsoft Research, is a peer produced sensor network that consists of sensors deployed by contributors across the globe. It is a unified system to collect, share, process, and query sensory data from the globally shared sensor network. It aims to provide an open architecture such that third parties can easily register sensors or repositories of sensory data to contribute as part of SenseWeb. Meanwhile, SenseWeb also enables third parties to easily develop sensing applications that use the shared sensing resources provided by SenseWeb.

One sample application of SenseWeb is **SensorMap**, developed by Microsoft Research. SensorMap mashes up sensor data from SenseWeb on a map interface, and provides interactive tools to selectively query sensors and visualize data, along with authenticated access to manage sensors.

1.2 Architecture
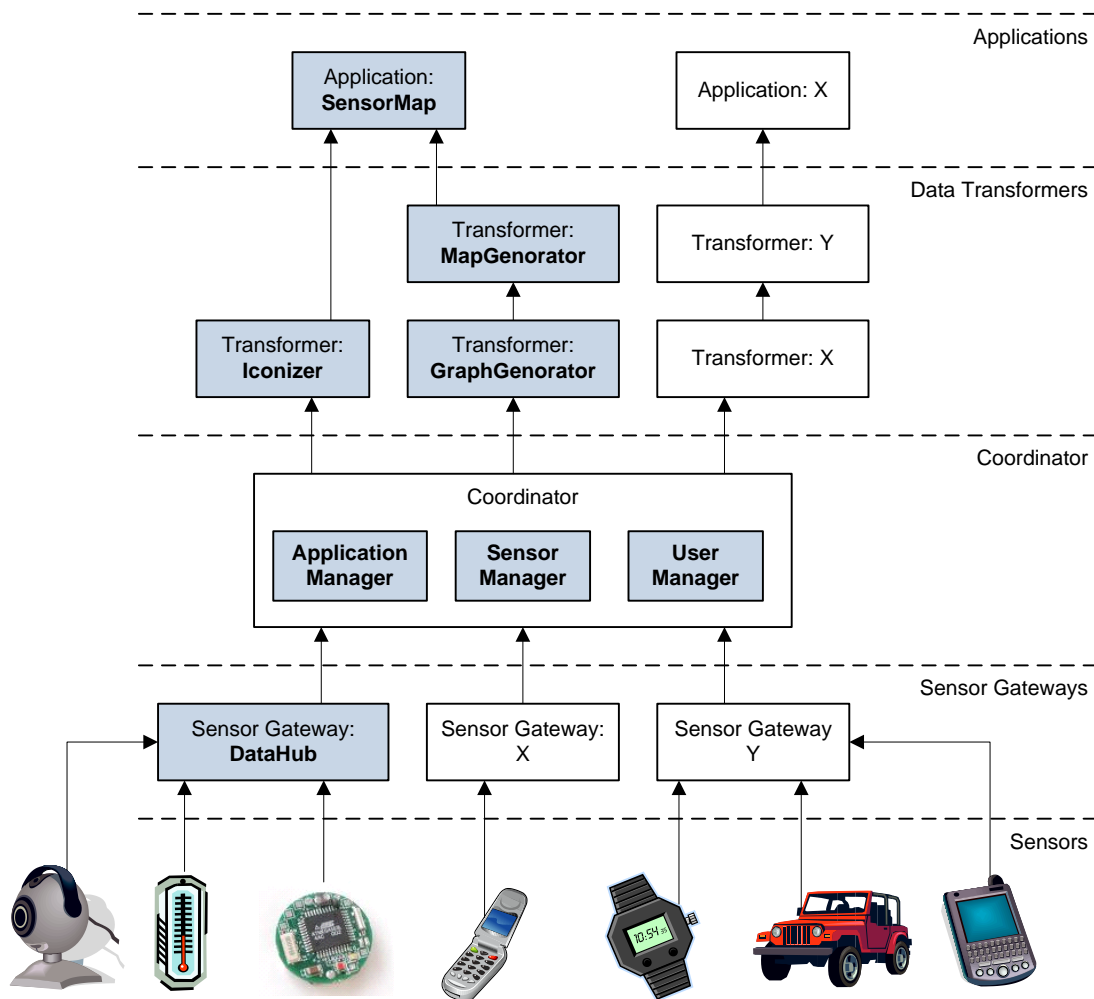
Figure 1 System architecture

SenseWeb has a layered architecture as depicted in Figure 1. The blue blocks are modules provided and deployed by Microsoft Research, while the white blocks indicate possible extensions by third parties. Next, we give an overview of all the modules starting from the narrow waist, Coordinator.

Coordinator

The **Coordinator** is the central point of access into the system for all applications and sensor contributors. It consists of three components: **User Manager**, **Sensor Manager**, and **Application Manager**. The **user manager** module implements user authentication mechanisms. Its web service URL is http://Sensormap.org/SenseWebV3/UserManager/Service.asmx.

The **sensor manager** acts as an index of all available sensors and their characteristics. It acts similar to a DNS server on the Internet, converting user friendly sensor descriptions such as location boundaries, logical names, or sensor types to physical sensor identifiers. Besides providing indexing service to upper layers, it also includes APIs for sensor gateways to manipulate both sensors and their types. More concretely, it allows definition of new sensor types, registration of new sensors of defined types, modification of characteristics of registered sensors, and deletion of registered sensors. Its web service URL is http://sensormap.org /SenseWebV3/SensorManager/Service.asmx.

The **application manager** serves as the major access point to shared data for upper layers. It manages the underlying data providers (i.e., sensor gateways). At the same time, it accepts sensing queries from upper layers and attempt to satisfy them based on available sensing resources. To minimize the load on the sensors or the respective sensor gateways, the application managers leverages overlaps among multiple application needs, and attempts to combine the requests for common data. It also caches recently accessed data so that future queries without stringent real-time requirements could be served by local caches. Its web service URL is http://sensormap.org/SenseWebV3/ApplicationManager/Service.asmx.

Sensor Gateways

The sensing resources form the foundation of the entire system. Sensors could be measuring various physical variables and may be static or mobile, carried by humans, on vehicles, or in robots.

Since sensors may be built using many different platforms widely varying in processing power, energy, and bandwidth capabilities, they may have different interfaces to access them. Low powered wireless sensor nodes may use 15.4 radios to communicate, while higher power and higher bandwidth sensors may need Firewire or similar interfaces. They may or may not be connected at all times. To hide much of this complexity, all sensors are connected to **Sensor Gateways** that provide a uniform interface to all components above it. The gateway implements sensor specific methods to communicate with the sensor. However, other components of SenseWeb access the gateway to obtain sensor data streams, to submit data collection demands, or access sensor characteristics through a standardized web service API. Each sensor contributor may maintain his or her own gateway. The gateway may also implement sharing policies defined by the contributor. For instance, the gateway may maintain all raw data in its local database, possibly for local applications run by the sensor owner but only make certain non-privacy-sensitive parts of the data or data at lower sampling rates, available to the rest of the SenseWeb.

We implemented one default gateway, called **Datahub**. DataHub may be used by sensor contributors who do not wish to maintain their own gateway. Individual sensors can publish their data to DataHub through a web service API. DataHub also has a user authentication module such that only authenticated users could access the services of DataHub. The web service URL of DataHub is http://www.sensormap.org/SenseWebV3/DataHub/Service.asmx. The web service URL for its user management module is: http://www.sensormap.org/SenseWebV3/UserManagerDH/Service.asmx.

Sensors

We do not define the interface between sensors and their gateways for third party implementations of gateways. However, for users who wish to use DataHub as their default gateway should implement wrappers for their sensors to provide web service APIs required by DataHub. Wrappers for common types of sensors including wireless motes and network cameras are available (http://research.microsoft.com/nec/msrsense/).

Data Transformers

The role of a **transformer** is to convert data semantics through processing. For example, a transformer may extract the people count from a video stream. Additional examples of data transformers are unit conversion, data fusion, and data visualization services. Moreover, application developers can extend SenseWeb's processing functionality by writing new transformers on top of Coordinator's primitive access methods. Domain experts may implement various transformers for different sensor data using suitable domain specific algorithms. One example of a transformer is the **Iconizer** implemented in our prototype: it converts raw sensor readings into an icon that represents sensor type in its shape and sensor value in its color. Graphical applications may use the output of this transformer instead of raw sensor values. Other examples are the **GraphGenerator** and the **MapGenerator**. The former obtains raw sensor readings and generates 2D spatial graphs, while the latter converts 2D spatial graphs to map tiles that could be directly overlaid on top of existing maps.

Applications

Applications are ultimate consumers of sensor data. These may be interactive applications where human users specify their data needs manually, such as a user queries for average hiker heart rate over the last season on a particular trail, or automated applications in backend enterprise systems that access sensor streams for business processing, such as an inventory management application that access shopper volume from parking counters, customer behaviors from video streams, and correlates them with sales records. One existing example is our **SensorMap** portal that visualizes sensors and their data on top of maps from Virtual Earth.

1.3 Security

For security concerns, in most cases, we only allow registered users to invoke the web services provided by us. To invoke a web method, you should first authenticate yourself by providing user name and password to the user manager module. The user manager module will then issue a globally unique identifier as your temporary pass code, which will expire after certain time period, say, 10 minutes. Before the expiration of the pass code, you can use your user name and pass code to invoke other web service APIs provided by the coordinator.

We implement the user manager as a separate web service, so that it could be deployed as an encrypted website in final deployments.

Similarly, inside our DataHub, we provide two major modules deployed as two web services. One is called DataHub, which manages sensors and data. The other is UserManagerDH, which has the same interface as the user manager within the coordinator. However, the group of users managed by Data-Hub is different from those managed by the coordinator.

Lesson 2 – Use DataHub: the Default Sensor Gateway

There are three types of sensors we currently support: **scalar sensors** that output a single numeric value a time, **binary sensors** that create binary data and **vector sensors** that generate an array of numeric values. Next, we will introduce how to manipulate each of the types by using the web service APIs of Data-Hub.

2.1 Initial Setup

The following steps are required for all the examples in this chapter. The instructions are for Visual Studio C#. We assume the readers have certain knowledge about C# and Visual Studio. Users using other languages should follow similar steps.

Step 1: Create new project

Open Visual Studio (or Visual C# Express), and create a new project, setting the project type to Visual C# and template to console application. In the following, we assume that the project is named **TestData-Hub.**

Step 2: Add web reference

1. Create a web reference to DataHub web service using the following URL
   http://sensormap.org/SenseWebV3/DataHub/Service.asmx
   and name it **DataHub.**
2. Create a web reference to User Manager web service using the following URL
3. http://sensormap.org/SenseWebV3/UserManagerDH/Service.asmx
   and name it **UserManagerDH**.

Use the following code to create service handlers to access web methods later:

```
DataHub.Service dataHub = new DataHub.Service();
UserManagerDH.Service userManagerDH = new UserManagerDH.Service();
```

Step 3: Authenticate yourself and get passcode

```
// authenticate yourself
string userName = "sknath@gmail.com";
string password = "yourPassword";
Guid passCode = userManagerDH.GetPassCode(userName, password);
```

Note: we did not include any exception handling in the examples.

2.2 Sensor Type Management

Example 2_2_1 Create a non-vector sensor type

Use the following code to create a new scalar type

```
string sensorTypeName = "yourSensorTypeName";
string sensorTypeUri = "yourSensorTypeUri";
string output = dataHub.CreateSensorType(userName, passCode,
sensorTypeName, sensorTypeUri);
Console.WriteLine(output);
```

After compiling and execution, you should expect to see "OK".

Example 2_2_2 Create a vector sensor type

Declare new type name:

```
string vectorSensorTypeName = "yourVectorSenosrTypeName";
```

Each vector type is essentially a combination of multiple non-vector types. We need to define the array of non-vector types by using their type names:

```
string[] components = new string[3] { "Temperature", "Humidity",
"Pressure" };
```

Finally, let's create the new vector type:

```
string output = dataHub.CreateVectorSensorType(userName, passCode,
vectorSensorTypeName, components);
Console.WriteLine(output);
```

After compiling and execution, you should expect to see "OK".

2.3 Sensor Management

Next we will show an example which first registers a sensor and fetches the details of this registered sensor by its name, then changes its location and fetches the details again, and finally deletes the sensor.

Example 2_3_1 Register, query, update, and delete a sensor

```
// this is a themometer sensor
string sensorTypeUri =
"http://research.microsoft.com/nec/sensor/type/SensorType.owl#Thermometer";
string sensorName = "yourSensorName";
string sensorURL = "yourSensorURL";
string sensorDescription = "yourSensorDescription";
double latitude = -1;
double longitude = -1;
double altitude = 0;

// decare a new sensor
DataHub.SensorInfo sensor = new DataHub.SensorInfo();
sensor.dataType = "scalar";
sensor.sensorType = sensorTypeUri;
sensor.publisherName = userName;
sensor.sensorName = sensorName;
sensor.url = sensorURL;
sensor.description = sensorDescription;
sensor.latitude = latitude;
sensor.longitude = longitude;
sensor.altitude = altitude;
```

Then you can register it:

```csharp
        // register the sensor
        string output = dataHub.InsertSensor(userName, passCode, sensor);
        Console.WriteLine(output);

        // get sensor description by name
        output = dataHub.GetSensorDescriptionByName(userName, passCode,
sensorName);
        Console.WriteLine(output);

        // get sensor description by name
        output = dataHub.GetSensorDescriptionByName(userName, passCode,
sensorName);
        Console.WriteLine(output);

        // update the location of the sensor
        double newLatitude = 0;
        double newLongitude = 0;
        double newAltitude = 0;
        output = dataHub.UpdateSensorLocation(userName, passCode,
sensorName, newLatitude, newLongitude, newAltitude);
        Console.WriteLine(output);

        // get sensor description by name
        output = dataHub.GetSensorDescriptionByName(userName, passCode,
sensorName);
        Console.WriteLine(output);

        // delete the sensor
        output = dataHub.DeleteSensor(userName, passCode, sensorName);
        Console.WriteLine(output);
```

After compiling and execution, you should expect to see the following output:

2.4 Data Management

Next example shows how we store and retrieve data from a sensor or sensors producing scalar data. In this example, we first register two sensors. Then we store and retrieve data for the first sensor to showcase how we can access data for a single sensor. Next, we store and retrieve data for both sensors at the same time since our interface also supports data access of multiple sensors in a batch. Finally, we delete the two sensors.

Example 2_4_1 Store and get scalar data for a single sensor or a batch of sensors

```csharp
// this is a themometer sensor
string sensorTypeUri =
"http://research.microsoft.com/nec/sensor/type/SensorType.owl#Thermometer";
string sensorURL = "yourSensorURL";
string sensorDescription = "yourSensorDescription";
double latitude = -1;
double longitude = -1;
double altitude = -1;

// decare and register sensor 1 named yourSensorName1
string sensorName1 = "yourSensorName1";
DataHub.SensorInfo sensor = new DataHub.SensorInfo();
sensor.dataType = "scalar";
sensor.sensorType = sensorTypeUri;
sensor.publisherName = userName;
sensor.sensorName = sensorName1;
sensor.url = sensorURL;
sensor.description = sensorDescription;
sensor.latitude = latitude;
sensor.longitude = longitude;
sensor.altitude = altitude;
string output = dataHub.InsertSensor(userName, passCode, sensor);
Console.WriteLine(output);

// decare and register sensor 2 named yourSensorName2
string sensorName2 = "yourSensorName2";
sensor.sensorName = sensorName2;
output = dataHub.InsertSensor(userName, passCode, sensor);
Console.WriteLine(output);

// store scalar data from Sensor 1 into DataHub
DataHub.SensorData sensorDataToStore = new DataHub.SensorData();
sensorDataToStore.Data = 11;
sensorDataToStore.DataType = DataHub.DataType.Scalar;
sensorDataToStore.Timestamp = DateTime.Now;
output = dataHub.StoreScalarData(userName, passCode, sensorName1,
sensorDataToStore);
Console.WriteLine(output);

// get the latest scalar data of Sensor 1
DataHub.SensorData sensorDataRetrieved =
dataHub.GetLatestScalarData(userName, sensorName1);
Console.WriteLine("Latest scalar data from Sensor 1: "
                  + sensorDataRetrieved.Data);
```

```csharp
            // store scalar data of multiple sensors (sensor 1 and sensor 2)
as a batch
            string[] userNameArray = {userName, userName};
            Guid[] passCodeArray = {passCode, passCode};
            string[] sensorNameArray = {sensorName1, sensorName2};
            DataHub.SensorData[] sensorDataArray = new DataHub.SensorData[2];
            sensorDataArray[0] = new DataHub.SensorData();
            sensorDataArray[0].Data = 12;
            sensorDataArray[0].DataType = DataHub.DataType.Scalar;
            sensorDataArray[0].Timestamp = DateTime.Now;
            sensorDataArray[1] = new DataHub.SensorData();
            sensorDataArray[1].Data = 22;
            sensorDataArray[1].DataType = DataHub.DataType.Scalar;
            sensorDataArray[1].Timestamp = DateTime.Now;
            output = dataHub.StoreScalarDataBatch(userNameArray,
passCodeArray, sensorNameArray, sensorDataArray);
            Console.WriteLine(output);

            // get the latest scalar data of multiple sensors as a batch
            DataHub.SensorData[] sensorDataArrayRetrieved =
dataHub.GetLatestScalarDataBatch(userNameArray, sensorNameArray);
            Console.WriteLine("Latest scalar data from Sensor 1 and 2: "
                        + sensorDataArrayRetrieved[0].Data
                        + "," + sensorDataArrayRetrieved[1].Data);

            // delete sensor 1
            output = dataHub.DeleteSensor(userName, passCode, sensorName1);
            Console.WriteLine(output);

            // delete sensor 2
            output = dataHub.DeleteSensor(userName, passCode, sensorName2);
            Console.WriteLine(output);
```
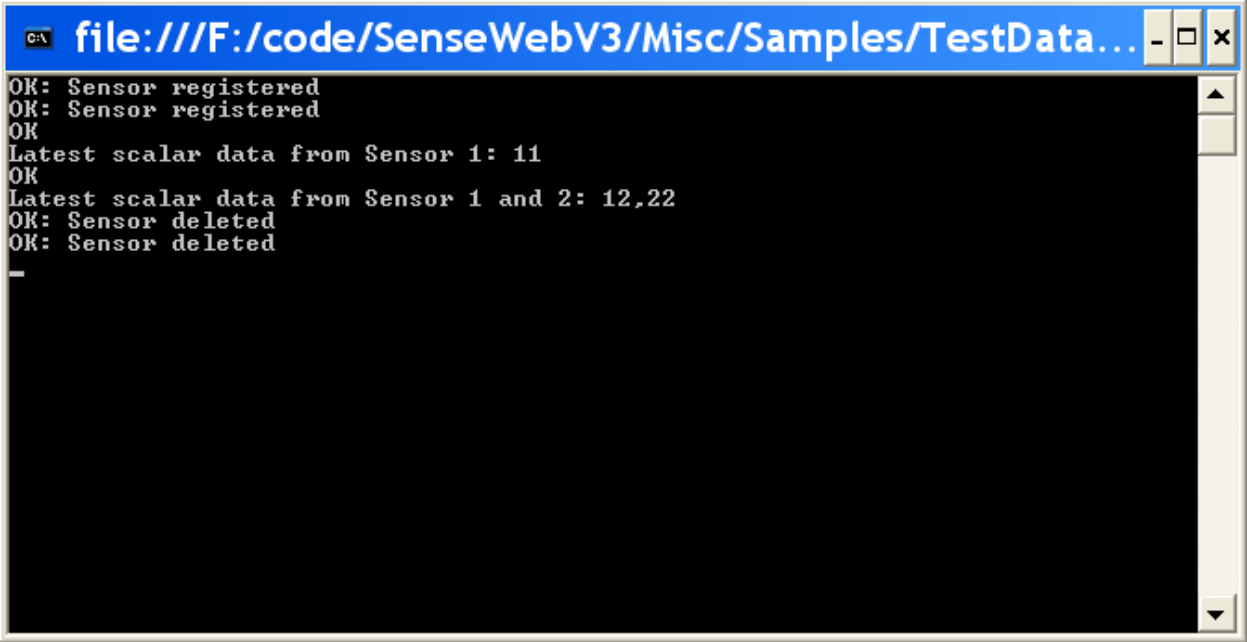
After compiling and execution, you should expect to see the following output:

file:///F:/code/SenseWebV3/Misc/Samples/TestData...

```
OK: Sensor registered
OK: Sensor registered
OK
Latest scalar data from Sensor 1: 11
OK
Latest scalar data from Sensor 1 and 2: 12,22
OK: Sensor deleted
OK: Sensor deleted
```

Lesson 3 - Implement Your Own Sensor Gateway

This lesson teaches how to implement your own sensor gateway, which manages a set of sensors and keeps a local repository of all the data from the set of sensors. We make the following assumptions about your gateway:

1. Your gateway should have mechanisms for your own users (managed by your gateway) to register sensor and publish data.

2. Sensors can be uniquely addressed using publisher name (the account name of the user who registered the sensor) and sensor name.

To plug your gateway into SenseWeb, there are a few steps you should take, which are to be detailed in this lesson. Some of the terminologies we use in this lesson can be found in **Reference – Terminologies**.

Step 1 – Apply for a user account of SenseWeb for your sensor gateway

Later, we will release the interface for user registration. For now, email sumann@microsoft.com to get your temporary account and password.

Step 2 – Register sensor types and sensors with SenseWeb

Initial Setup

4. Create a web reference to Sensor Manager web service using the following URL
http://sensormap.org/SenseWebV3/SensorManager/Service.asmx
and name it **SensorManager.**
5. Create a web reference to User Manager web service using the following URL
http://sensormap.org/SenseWebV3/UserManager/Service.asmx
and name it **UserManager**.

Register sensor types and sensors

If you have any special sensor which cannot be categorized into one of our predefined [primary sensor types](), you should register a new sensor type with SenseWeb. If you have any new [vector sensor types]() which have not been defined by other users, you should register it with SenseWeb. For any sensors you plan to serve data for SenseWeb, you should register them with SenseWeb.

The following sample code shows how to register a non-vector sensor type, how to register a vector sensor type, and how to register a new sensor. Before you invoke any service of SenseWeb, you need to authenticate yourself and get a pass code which will be valid for about 10 minutes. At the end of the example, the registered sensor is deleted to keep the system clean.

Example 3_1 Register sensor types and sensors

```
// create an accessor to web services of SensorManager
SensorManager.Service sensorManager = new SensorManager.Service();

/*
```

```csharp
                 * authenticate yourself with SenseWeb and get a temporary
passCode (expire in 10 minutes)
             */
string userName = "yourUserName";
string passwd = "yourPasswd";
UserManager.Service userManager = new UserManager.Service();
Guid passCode = userManager.GetPassCode(userName, passwd);

/*
* register a non-vector sensor type
*/
string sensorTypeName = "yourSensorTypeName";// say Thermometer
string dataType = "scalar"/"image"/"html";
string unit = "cm";
string iconUrl = "icon.jpg";
string output = sensorManager.CreateSingularSensorType(userName,
                   passCode, sensorTypeName,dataType, unit, iconUrl);
 Console.WriteLine(output);

/*
* register a non-vector sensor
*/

// assume this is a themometer sensor
string sensorType = "Thermometer"; //Thermometer is an existing sensortype.
string sensorName = "yourSensorName";
string latitude = "-1";
string longitude = "-1";
string altitude = "0";
string desc =  "desc";  //some description about the sensor
string dataType = "scalar"; // image, html etc
string keyWords = "keywords for the sensor";
string wsURL = "web service URL that provides data to the sensor";
string groupName = "default"; // the group that this sensor will belongs to;
string accessControl = "public"; // private/protected

sensorManager.RegisterSingularSensor(userName,passCode,sensorName,sensorType,
altitude, longitude, latitude, desc, dataType, keyWords, wsURL, groupName,
accessControl)

/*
* register a vector sensor.
*/

string vectorSensorType = "existing type"; // or new type,if new type then it
will be created automatically.
string dataType = "vector";
string icon = "icon.jpg"; // where the icon should be uploaded using the
utility provided with Sensormap.org
string paramTypes = "Temperature|Pressure"; //sensortypes with pipe seperated.
string paramNames = "Name to be displayed for each parameter sensors"//should
be seperated with pipe symbol. For Eg: "Temp Sensor|Pressure Sensor"
string paramComments = "comments for each parameter sensors"// for eg:
"comment1|comment2"
string paramMetaData = "metadata for the parameter sensors"//For eg:
"Medium:Air:,Height:10:cm|Medium:Water:,Height:-100:m"     where    for    each
parameter the values should be seperated with '|' and for each metadata it
```

```
should be seperated with ',' and each key,value,unit pair should be seperated
with ':'

sensorManager.RegisterCompositeSensor(userName, passCode, sensorName,
vectorSensorType, altitude, longitude, latitude, desc, dataType, keyWords,
wsURL, groupName, accessControl, icon, paramTypes, paramNames, paramComments,
paramMetaData)

// get sensor description by name
output = sensorManager.GetSensorByPublisherAndName(userName, passCode,
sensorName);
Console.WriteLine(output);

// update the location of the sensor
double newLatitude = 0;
double newLongitude = 0;
double newAltitude = 0;
output = sensorManager.UpdateSensorLocation(publisherName, passCode,
originalPublisherName, sensorName, newLatitude, newLongitude, newAltitude);
Console.WriteLine(output);

// delete the sensor
output = sensorManager.DeleteSensor(userName, passCode, sensorName);
Console.WriteLine(output);
```

Note that you should replace the fake `yourUserName` and `yourPasswd` in the code with your real username and password to make it work. Assuming the code is inserted into a console application, after compiling and execution, you should see the following output:

Step 3 – Implement a web service interface to serve data for SenseWeb

List of APIs
The web service interface should expose the following list of web methods:

1. GetLatestBinarySensorData
   Returns the latest image data reported by a sensor

   ```
   SensorData GetLatestBinarySensorData(string publisherName, string
   sensorName, DataType type)
   ```

2. GetLatestScalarData
   Gets the latest data published by a sensor

   ```
   SensorData GetLatestScalarData(string publisherName, string
   sensorName)
   ```

3. GetLatestScalarDataBatch
   Gets latest data published by a set of sensors

   ```
   SensorData[] GetLatestScalarDataBatch(string[] publisherNames,
   string[] sensorNames)
   ```

4. [GetScalarData](#)
   Gets the data published by a sensor within a specified time window

   ```
   SensorData GetScalarData(string publisherName, string sensorName,
   DateTime startTime, DateTime endTime)
   ```

5. [GetVectorComponentData](#)
   Gets data from one component sensor of a vector sensor

   ```
   SensorData GetVectorComponentData(string publisherName, string
   sensorName, int componentIndex)
   ```

6. [GetVectorData](#)
   Gets data from a vector sensor

   ```
   SensorData[] GetVectorData(string publisherName, string sensorName,
   string vectorType)
   ```

Data Types

Two special data types are defined by us:

```
public enum DataType
{
    Unknown = 0,
    Scalar = 1,
    BMP = 2,
    JPG = 3,
    GIF = 4,
    VECTOR = 5,
    HTML = 6
}
```

```
public class SensorData
{
    public DateTime Timestamp;
    public DateTime[] Timeseries;
    public object Data;
    public int SensorType;
    public string SensorTypeString;
    public DataType DataType;
}
```

For scalar sensors, the `Data` field of `SensorData` should be `Double`, and the `Timestamp` field stores the timestamp; for binary sensors, the data type of `Data` should be `byte[]`; and for vector sensors, the data type of `Data` should be `Double[]` and timestamps are stored in `Timeseries` (**Note**: the sequence of both `Data` and `Timeseries` should be correspondent to the sequence of component sensors of the vector sensor type).

Step 4 – Deploy your web service

You should deploy the implemented web service onto a public website. To integrate your gateway with SenseWeb, you have to take the final step to register it with SenseWeb. We will release the gateway

registration interface later. For now, please email the URL of your web service to su-mann@microsoft.com for manual registration.

Lesson 4 – Build Applications upon SenseWeb

SenseWeb serves as the global data indexing engine, from which you should be able to pull data out from any connected gateways. The lesson introduces code examples on how to pull data out from SenseWeb and build application upon the data. For example, you could get all the temperature data of a specified geographical region and draw contour maps based on the data. Here are the steps to access data from SenseWeb.

4.1 Initial Setup

1. Create a web reference to Sensor Manager web service using the following URL
   http://sensormap.org/SenseWebV3/ApplicationManager/Service.asmx
   and name it **ApplicationManager.**
2. Create a web reference to User Manager web service using the following URL
   http://sensormap.org/SenseWebV3/UserManager/Service.asmx
   and name it **UserManager**.

4.2 Query sensors and sensor data

The following example shows how to get a list of sensors within certain geographical boundaries and of certain types, and how to get data for a known list of sensors either from the default MSR DataHub or a specific user defined gateway.

Example 4_2_1 Query sensors and sensor data

```
            // create an accessor to web services of ApplicationManager
            ApplicationManager.Service appManager = new
ApplicationManager.Service();

            /*
             * authenticate yourself with SenseWeb and get a temporary
passCode (expire in 10 minutes)
             */
            string userName = "yourUserName";
            string passwd = "yourPasswd";
            UserManager.Service userManager = new UserManager.Service();
            string passCode = userManager.GetPassCode(userName,
passwd).ToString();

            /*
             * Get the list of sensors within a polygon
             */
            // create the list of points that define a polygon
            ApplicationManager.PointF[] points = new
ApplicationManager.PointF[4];
            points[0] = new ApplicationManager.PointF();
            points[0].lat = 47.00;
            points[0].lon = -122.00;
            points[1] = new ApplicationManager.PointF();
            points[1].lat = 47.00;
            points[1].lon = -123.00;
            points[2] = new ApplicationManager.PointF();
            points[2].lat = 48.00;
```

```csharp
                points[2].lon = -123.00;
                points[3] = new ApplicationManager.PointF();
                points[3].lat = 48.00;
                points[3].lon = -122.00;

                // we are interested in traffic sensors
                ApplicationManager.SensorTypeEnum[] types = new
ApplicationManager.SensorTypeEnum[1];
                types[0] = ApplicationManager.SensorTypeEnum.Traffic;

                ApplicationManager.ResponseOfArrayOfSensorInfo sensorInforsOutput
= appManager.FindSensorsByLocation(userName, passCode, points, "", types);
                if (sensorInforsOutput != null)
                {
                    Console.WriteLine(sensorInforsOutput.message);
                    if (sensorInforsOutput.returnData != null
                    && sensorInforsOutput.returnData.Length > 0)
                    {
                        int i = 1;
                        foreach (ApplicationManager.SensorInfo sensor in
sensorInforsOutput.returnData)
                        {
                            Console.WriteLine("Sensor " + i + ":"
                                            + sensor.sensorName + ","
                                            + sensor.publisherName + ";");
                            i++;
                        }
                    }
                }

                /*
                 * Get the most recent data of the list of sensors
                 */
                if (sensorInforsOutput != null
                    || sensorInforsOutput.returnData != null
                    || sensorInforsOutput.returnData.Length != 0) {

                    ApplicationManager.SensorInfo[] sensors =
sensorInforsOutput.returnData;
                    string[] publisherNames = new string[sensors.Length];
                    string[] sensorNames = new string[sensors.Length];
                    for (int i = 0; i < sensors.Length; i++)
                    {
                        sensorNames[i] = sensors[i].sensorName;
                        publisherNames[i] = sensors[i].publisherName;
                    }

                    /*
                     * Get data from MSR DataHub
                     */
                    ApplicationManager.ResponseOfArrayOfSensorData
sensorDatasOutput = appManager.CollectDataSnapshotScalar(userName, passCode,
publisherNames, sensorNames);
                    if (sensorDatasOutput != null)
                    {
                        Console.WriteLine(sensorDatasOutput.message);
                        if (sensorDatasOutput.returnData != null
```

```
                          && sensorDatasOutput.returnData.Length > 0)
              {
                  ApplicationManager.SensorData[] sData =
sensorDatasOutput.returnData;
                  int i = 1;
                  foreach (ApplicationManager.SensorData data in sData)
                  {
                      Console.WriteLine("Data " + i + ":"
                                        + data.Timestamp
                                        + "," + data.Data);
                      i++;
                  }
              }
          }

          /*
           * Get data from a specific gateway
           */
          sensorDatasOutput =
appManager.CollectDataSnapshotScalarGW(userName, passCode, publisherNames,
sensorNames, "http://sensormap.org/SenseWebV3/DummyDataHub/Service.asmx");
          if (sensorDatasOutput != null)
          {
              Console.WriteLine(sensorDatasOutput.message);
              if (sensorDatasOutput.returnData != null
                  && sensorDatasOutput.returnData.Length > 0)
              {
                  ApplicationManager.SensorData[] sData =
sensorDatasOutput.returnData;
                  int i = 1;
                  foreach (ApplicationManager.SensorData data in sData)
                  {
                      Console.WriteLine("Data " + i + ":"
                                        + data.Timestamp
                                        + "," + data.Data);
                      i++;
                  }
              }
          }
      }
```

Note that you should replace the fake `yourUserName` and `yourPasswd` in the code with your real username and password to make it work. Assuming the code is inserted into a console application, after compiling and execution, you should see the following output:
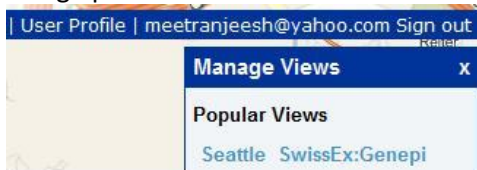
```
Success: 9 sensors found.
Sensor 1:ES-541R:MME_Stn,admin;
Sensor 2:ES-123D:_MN_Stn,admin;
Sensor 3:ES-137R:MMN_Stn,admin;
Sensor 4:ES-139R:_RN_Stn,admin;
Sensor 5:ES-538R:MME_Stn,admin;
Sensor 6:ES-533D:_ME_Stn,admin;
Sensor 7:ES-533D:_MW_Stn,admin;
Sensor 8:ES-528D:_ME_Stn,admin;
Sensor 9:ES-540R:MMW_Stn,admin;
Data received from 9 sensors.
Data 1:1/3/2008 11:32:07 PM,3.5
Data 2:1/3/2008 11:32:07 PM,19.6666666666667
Data 3:1/3/2008 11:32:07 PM,10.9166666666667
Data 4:1/3/2008 11:32:07 PM,0
Data 5:1/3/2008 11:32:07 PM,7.58333333333333
Data 6:1/3/2008 11:32:08 PM,8.91666666666667
Data 7:1/3/2008 11:32:08 PM,4.25
Data 8:1/3/2008 11:32:08 PM,6.41666666666667
Data 9:1/3/2008 11:32:08 PM,6.5
Data received from 2 sensors.
Data 1:1/3/2008 11:32:09 PM,11.11
Data 2:1/3/2008 11:32:09 PM,22.22
-
```

Lesson 5 –SensorMap GUI Features

5.1 Creating Groups and associating users to groups

If you are a new user, create a login with SensorMap application and send a request mail to senseweb@microsoft.com to activate the account. Once you have an activated account, log on to SensorMap and find the user profile menu appeared on the menu items. Click on the menu and you can

a) Create Group/Make it as a child of existing group.
b) Delete Group
c) Add users to group (Only owner of the group can add users to the group, so that the pro-tected sensors of that group can be viewed by the group members.). The user account to be added should be an activated account.
d) Delete users from group
e) Change password.

5.2 Register Sensors

If you are a new user, create a login with SensorMap application and send a request mail to senseweb@microsoft.com to activate the account. Once you have an activated account, log on to SensorMap first, before you add a sensor.

a) Right click on Sensormap to get menu items.
b) Click on Register Sensor menu
c) Select the type of sensor to be registered.
d) Add your SensorName & SensorType (for eg: SensorName-> Traffic1, SensorType-> Traffic)
e) Latitude & Longitude will be populated depends upon where you have clicked on the map. If you want you can change those values.
f) If the sensor implemented is not at sea level, you can change the altitude as well, so that with 3D the altitude will be displayed.
g) WebService URL should be the address of the web service from where you get data for the sensor, when hover over the sensor.
h) Group should be selected, if you want to protect/private the sensor only for that group then make the Access Control to "protected/private", otherwise it is going to be public.
Public : both sensor and sensor data are visible.
Protected : Only sensor would be displayed and not data until the authenticate users are logged in.
Private : Both sensor and sensor data will not be displayed until the users are logged in.
i) Fill the necessary details and click on "Create/Register Sensor"

5.3 Delete Sensors

a) The owner of the Sensors would see a "Delete" link on the sensor details, when they hover over the sensor (User should be logged in before). Refresh the page to see the effect.
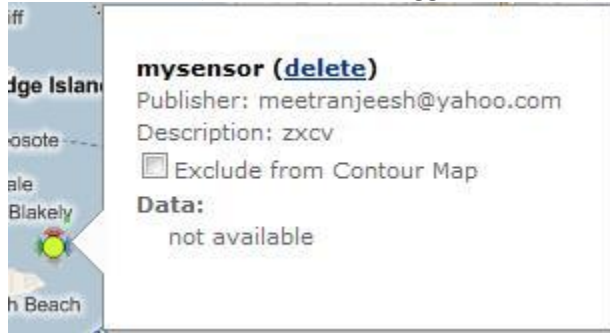
5.4 Embedding Sensor Map

a) Right click on the map and select "Generate iFrame Url"
b) All fields are populated depends upon where you have clicked on the map, what is the current zoom level, what should be the size of the Frame etc.
c) All pre populated values can be changed.
d) Click on Display frame will show a miniature version of SensorMap application with sensors.
e) If the user is logged in, then he/she must be able to see even the protected sensors as well.
f) Click on other links to get the URL of the embed SensorMap url.
g) **With IE, user must be able to copy the iFrame Tag/iFrame URL, so that he/she can place in their webpages.**
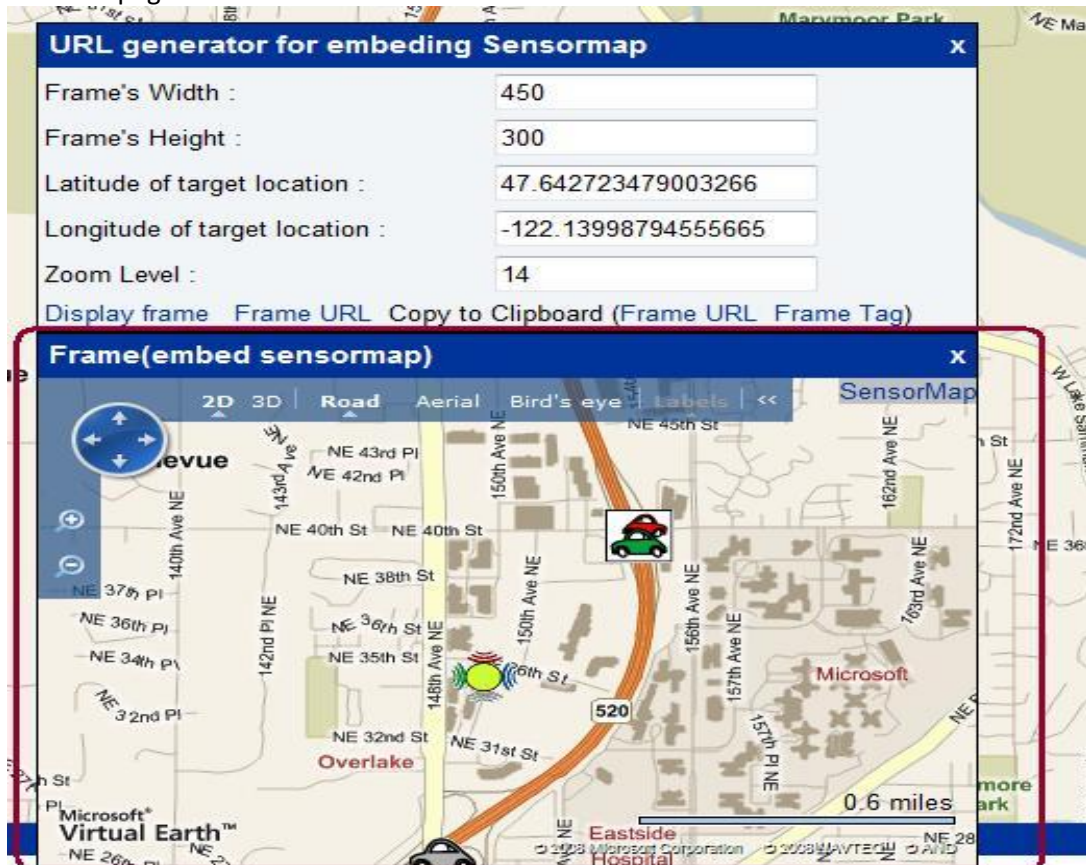
5.5 Chart, Download data
a) When hover over the sensor, data for the sensor would be displayed with a 'chart' link.
b) Click on the same would display chart data (graph) for a time span.
c) You could see 2 links "Download Data Text/Excel" to download data for the sensor that are displayed with the chart, either in .txt format or on .xls format.
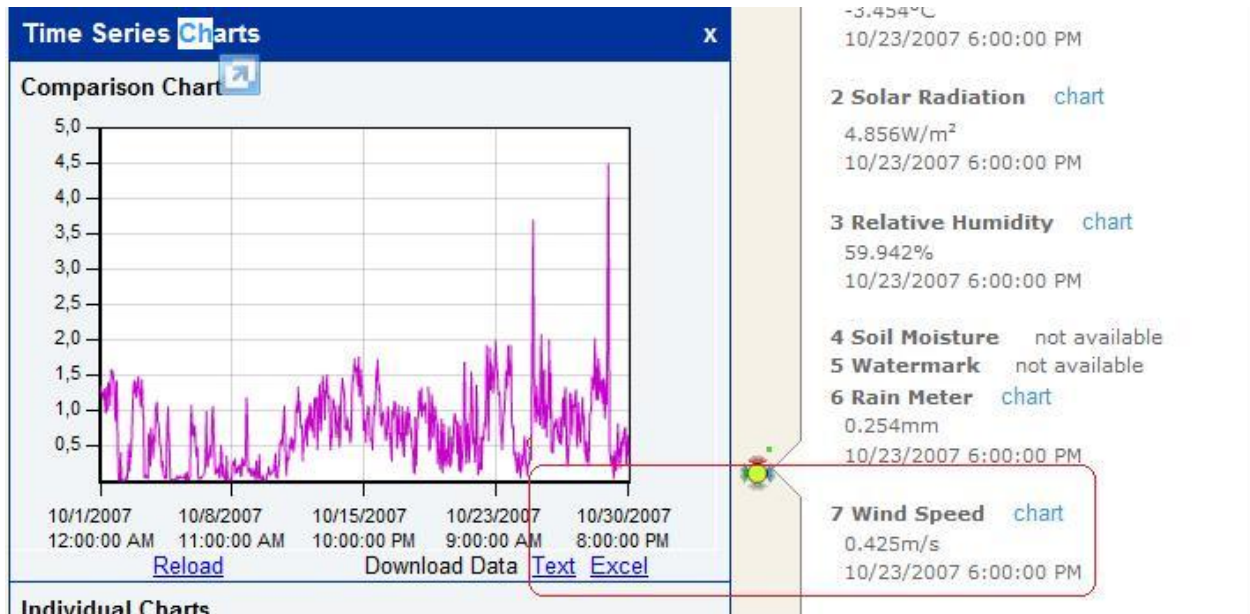
5.6 Filter Sensor by Login
Using the check box, the user can filter sensors that he created (or the group that he belongs to is created).

5.7 Set Sensor Data
If the user implements SetSensorData() with their datahub webservice, then the values for the sensor parameter can be resettled.
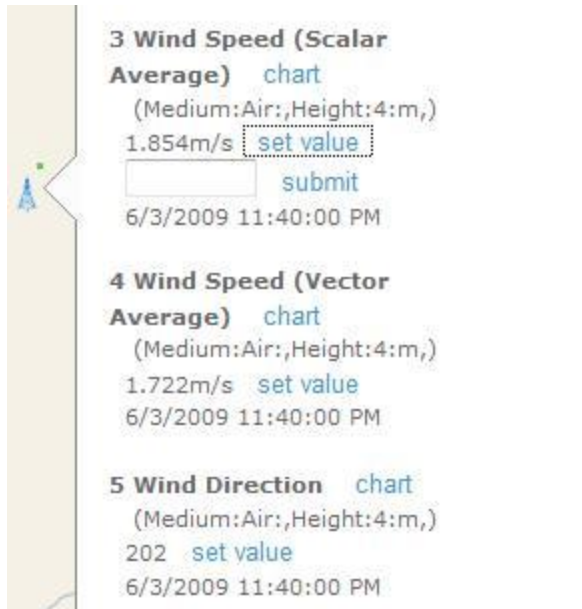
Reference –Terminologies

Sensor Types

We categorize sensors into different types based on logical meaning (e.g., temperature, humidity, motion, or video camera) and data structure (e.g., scalar data, binary data, or scalar/binary data arrays). In other words, sensors belonging to one sensor type should have the same logical meaning and data structure.

Non-vector Sensors

Sensors that generate only a single data value at a time instance are called non-vector sensors. For example, a thermometer that generates one temperature reading at a time is a non-vector sensor.

Vector Sensors

Sensors that may generate multiple data values at a time instance since it is equipped with multiple sensor modules. For example, a MicaZ mote that can generate temperature, light, and magnetic readings is a vector sensor.

Primary Sensor Types

We already defined a list of predefined types for non-vector sensors, including **Unknown**, **Generic**, **Temperature**, **Video**, **Traffic**, **Parking**, **Pressure**, and **Humidity**. User can register new types for non-vector sensors through SenseWeb as well. Both the predefined types and the user-defined types for non-vector sensors are called primary sensor types.

Vector Sensor Types

User can also register vector sensor types as an array of primary sensor types.

Scalar Sensors

One type of non-vector sensors whose output data is numeric. One example is a thermometer outputing the current temperature value.

Binary Sensors

One type of non-vector sensors whose output data is non-numeric. One example is a video camera which generates an image every 5 seconds.

Vector Sensors

Sensors that may generate multiple data values (binary or scalar) at a time. Essentially, each vector sensor is an array of non-vector sensors. Currently, we only support vector sensors that consists only scalar sensors.

Component Sensor

One vector sensor is essentially an array of non-vector sensors. Each of such non-vector sensors is called a component sensor of the vector sensor.

Reference – Web Service APIs

This lesson gives detailed descriptions of APIs of individual web services of SenseWeb.

2.1 Coordinator

User Manager Web Service API

Functionalities
- User authentication

URL

http://sensormap.org/SenseWebV3/UserManager/Service.asmx

List of APIs

GetPassCode
Authenticates a user and gets a token that he can use in place of password for certain period.

API Details
- Guid GetPassCode(
      string userName,
      string password)

Sensor Manager Web Service API

Functionalities
- Creation of new sensor types (single value types and vector types)

- Registration, modification, and deletion of sensors and their characteristics (i.e., sensor metadata)
- Query sensor metadata based on name, publisher, and geographical boundaries

URL

http://sensormap.org/SenseWebV3/SensorManager/Service.asmx

List of APIs

CreateSingularSensorType
dynamically creates a sensor type

RegisterSingularSensor
Registers a non composite sensor.

RegisterCompositeSensor
Registers a new composite sensor

DeleteSensor
Deletes an existing sensor

DeleteVectorSensor
Deletes an existing sensor

GetSensorDescriptionByName
Retrieves metadata of an existing sensor

GetSensorsByPublisher
Returns metadata of all the sensors published by a given publisher

SensorsInsidePolygon
Returns all the sensors within a polygon. Use the null to ignore some parameter.

UpdateSensorLocation
Modifies location of a sensor

API Details

- string CreateSingularSensorType(
        string publisherName,
        Guid passCode,
        string name,
        string dataType,
        string unit,
        string iconUrl)

    string RegisterSingularSensor(
        string publisherName,

```
                Guid passCode,
                string sensorName,
                string sensorType,
                string alt,
                string lon,
                string lat,
                string desc,
                string dataType,
                string keyWords,
                string wsURL,
                string groupName,
                string accessControl)


        string RegisterCompositeSensor(
                string publisherName,
                Guid passCode,
                string vectorSensorName,
                string vectorSensorType,
                string alt,
                string lon,
                string lat,
                string desc,
                string dataType,
                string keyWords,
                string wsURL,
                string groupName,
                string accessControl,
                string icon,
                string paramTypes,
                string paramNames,
                string paramComments,
                string paramMetaData)

•    string DeleteSensor(
                string publisherName,
                Guid passCode,
                string sensorName)


•    string DeleteVectorSensor(
                string publisherName,
                Guid passCode,
                string sensorName,
                string sensorType)


•    string GetSensorDescriptionByName(
                string publisherName,
                Guid passCode,
                string sensorName)


•    SensorInfo[] GetSensorsByPublisher(
```

```
            string publisherName,
            Guid passCode)


•   SensorInfo[] SensorsInsidePolygon(
            PointF[] polygon,
            PointF[] viewport,
            string searchStr,
            SensorTypeEnum[] sensorTypes)


•   string UpdateSensorLocation(
            string publisherName,
            Guid passCode,
            string sensorName,
            double latitude,
            double longitude,
            double altitude)
```

Application Manager Web Service API

Functionalities
- Registration of transformers
- Query sensor metadata based geographical boundaries, sensor types, and search terms
- Query latest snapshot data from scalar sensors hosted by DataHub or any specified gateway

URL
http://sensormap.org/SenseWebV3/ApplicationManager/Service.asmx

List of APIs

CollectDataSnapshotScalar
Obtain latest snapshot data from scalar sensors. Only sensors hosted by MSR DataHub are accessed.

CollectDataSnapshotScalarGW
Obtain latest snapshot data from scalar sensors, hosted by one specified gateway.

FindSensorsByLocation
Returns unique IDs and associated metadata for sensors available within requested polygonal region.

RegsiterTransformer
Make your transformer known to SenseWeb

API Details
```
•   SWLib.Response<DataHubWS.SensorData[]> CollectDataSnapshotScalar(
            string userID,
```

```
        string token,
        string[] publisherNames,
        string[] sensorNames)
```

- `SWLib.Response<DataHubWS.SensorData[]> CollectDataSnapshotScalarGW(`
```
        string userID,
        string token,
        string[] publisherNames,
        string[] sensorNames,
        string gatewayURL)
```

- `SWLib.Response<SWLib.SensorInfo[]> FindSensorsByLocation(`
```
        string userID,
        string token,
        SWLib.PointF[] polygon,
        string searchTerms,
        SWLib.SensorTypeEnum[] sensorTypes)
```

- `string RegsiterTransformer(`
```
        string userID,
        string token,
        string TxID,
        string TxGeoRSSUrl,
        string TxWsdlUrl,
        string TxGuiUrl,
        string description)
```

2.2 DataHub

DataHub Web Service API

Functionalities
- Sensor type management
  - Creation of new sensor types (single value types and vector types).
- Sensor metadata management
  - Registration, modification, and deletion of sensors and their characteristics (i.e., sensor metadata)
  - Query sensor metadata based on name, publisher, and geographical boundaries
- Sensor data management
  - Store data from a scalar sensor, data from a binary sensor, one component data of a vector sensor, or all data of a vector sensor into DataHub
  - Get latest data reported by a scalar sensor, a batch of scalar sensors, a binary sensor, or a vector sensor

URL
http://sensormap.org/SenseWebV3/DataHub/Service.asmx

List of APIs

CreateSensorType
Dynamically creates a new sensor type

CreateVectorSensorType
Dynamically creates a new vector sensor type

DeleteSensor
Deletes an existing sensor

DeleteVectorSensor
Deletes an existing sensor

GetLatestBinarySensorData
Returns the latest image data reported by a sensor

GetLatestScalarData
Gets the latest data published by a sensor

GetLatestScalarDataBatch
Gets latest data published by a set of sensors

GetScalarData
Gets the data published by a sensor within a specified time window

GetSensorDescriptionByName
Retrieves metadata of an existing sensor

GetSensorsByPublisher
Returns metadata of all the sensors published by a given publisher

GetVectorComponentData
Gets data from one component sensor of a vector sensor

GetVectorData
Gets data from a vector sensor

InsertSensor
Registers a new sensor

InsertVectorSensor
Registers a new sensor

SensorsInsidePolygon
Returns all the sensors within a polygon. Use the null to ignore some parameter.

[StoreBinaryData](#)
Send binary sensor data such as images, sound or video. Data are treated as a binary file. Time parameter is the time stamp of the first data.

[StoreScalarData](#)
Stores a sensor data in DataHub

[StoreScalarDataBatch](#)
Stores a series of sensor data in DataHub

[StoreVectorComponentData](#)
Stores data from one component sensor of a vector sensor

[StoreVectorData](#)
Stores data from a vector sensor

[UpdateSensorLocation](#)
Modifies location of a sensor

API Details

- ```
  string CreateSensorType(
        string publisherName,
        Guid passCode,
        string sensorType,
        string uri)
  ```

- ```
  string CreateVectorSensorType(
        string publisherName,
        Guid passCode,
        string vectorType,
        string[] componentTypes)
  ```

- ```
  string DeleteSensor(
        string publisherName,
        Guid passCode,
        string sensorName)
  ```

- ```
  string DeleteVectorSensor(
        string publisherName,
        Guid passCode,
        string sensorName,
        string sensorType)
  ```

- ```
  SensorData GetLatestBinarySensorData(
        string publisherName,
        string sensorName,
        DataType type)
  ```

- ```
  SensorData GetLatestScalarData(
  ```

```
            string publisherName,
            string sensorName)

•   SensorData[] GetLatestScalarDataBatch(
            string[] publisherNames,
            string[] sensorNames)

•   SensorData GetScalarData(
            string publisherName,
            string sensorName,
            DateTime startTime,
            DateTime endTime)

•   string GetSensorDescriptionByName(
            string publisherName,
            Guid passCode,
            string sensorName)

•   SensorInfo[] GetSensorsByPublisher(
            string publisherName,
            Guid passCode)

•   SensorData GetVectorComponentData(
            string publisherName,
            string sensorName,
            int componentIndex)

•   SensorData[] GetVectorData(
            string publisherName,
            string sensorName,
            string vectorType)

•   string InsertSensor(
            string publisherName,
            Guid passCode,
            SensorInfo sensorInfo)

•   string InsertVectorSensor(
            string publisherName,
            Guid passCode,
            SensorInfo sensorInfo)

•   SensorInfo[] SensorsInsidePolygon(
            PointF[] polygon,
            PointF[] viewport,
            string searchStr,
            SensorTypeEnum[] sensorTypes)

•   string StoreBinaryData(
            string publisherName,
            Guid passCode,
            string sensorName,
            SensorData data)
```

- ```
  string StoreScalarData(
       string publisherName,
       Guid passCode,
       string sensorName,
       SensorData data)
  ```

- ```
  string StoreScalarDataBatch(
       string[] publisherNames,
       Guid[] passCodes,
       string[] sensorNames,
       SensorData[] data)
  ```

- ```
  string StoreVectorComponentData(
       string publisherName,
       Guid passCode,
       string sensorName,
       int componentIndex,
       SensorData data)
  ```

- ```
  string StoreVectorData(
       string publisherName,
       Guid passCode,
       string sensorName,
       SensorData data)
  ```

- ```
  string UpdateSensorLocation(
       string publisherName,
       Guid passCode,
       string sensorName,
       double latitude,
       double longitude,
       double altitude)
  ```

- ```
  string SetSensorData(
       string publisherName,
       Guid passCode,
       string sensorName,
       string value,
       DateTime reportedtime)
  ```

User Manager Web Service API

Functionalities

- Authenticate users

URL

http://sensormap.org/SenseWebV3/UserManagerDH/Service.asmx

List of APIs

GetPassCode
Authenticates a user and gets a token that he can use in place of password for certain period.

API Details

- Guid GetPassCode(
      string userName,
      string password)