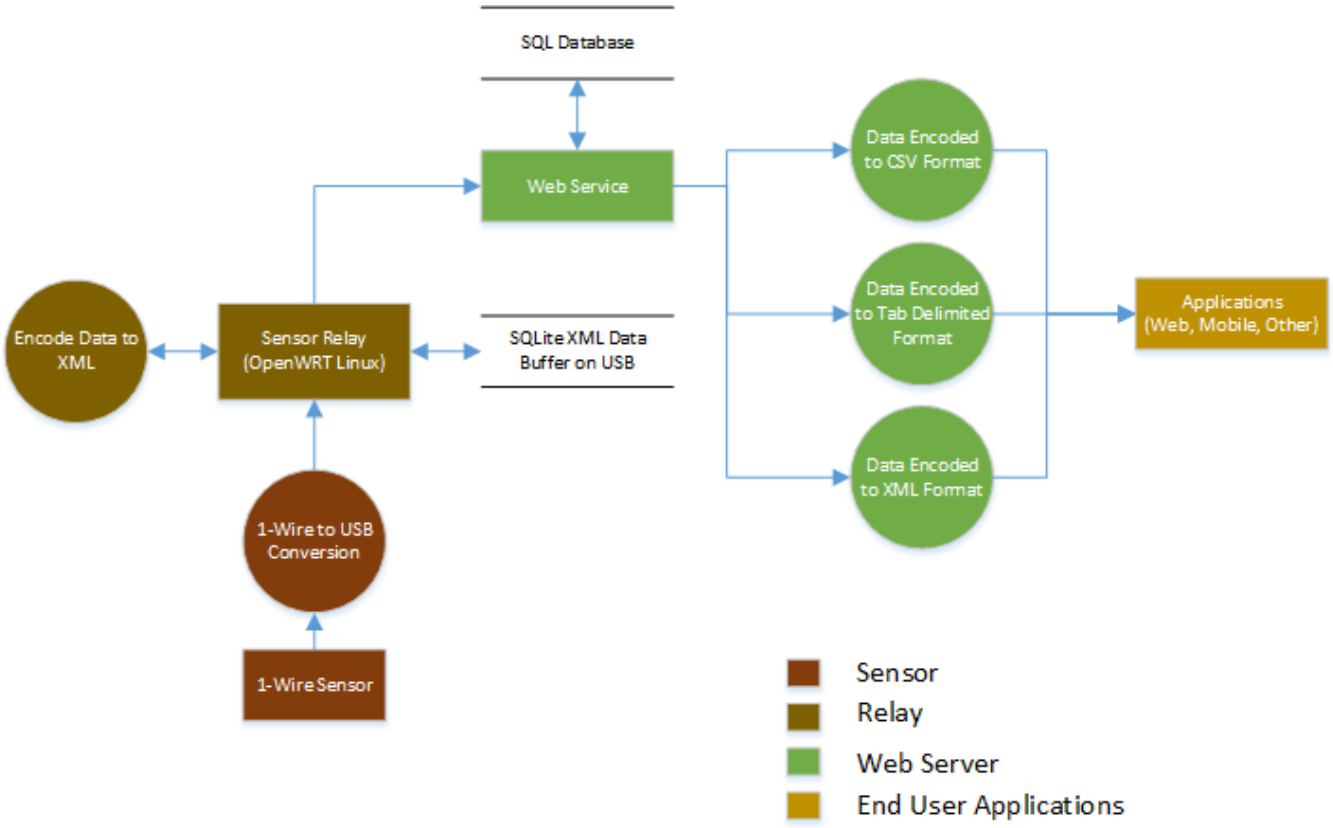
BigSense and LtSense

BigSense and LtSense are two application designed for storing sensor data and transmitting sensor data respectively. Although designed to work with each other, they can be used individually and extended to support other sensor types, data formats and storage services. The following diagram shows the data flow from embedded devices with attached 1-wire sensors. LtSense runs on the embedded device, which in this example, is an OpenWRT Linux system. It transmits sensor data at a per-configured sample rate to the BigSense web service, which can then offer the data, in various formats, to other web applications.

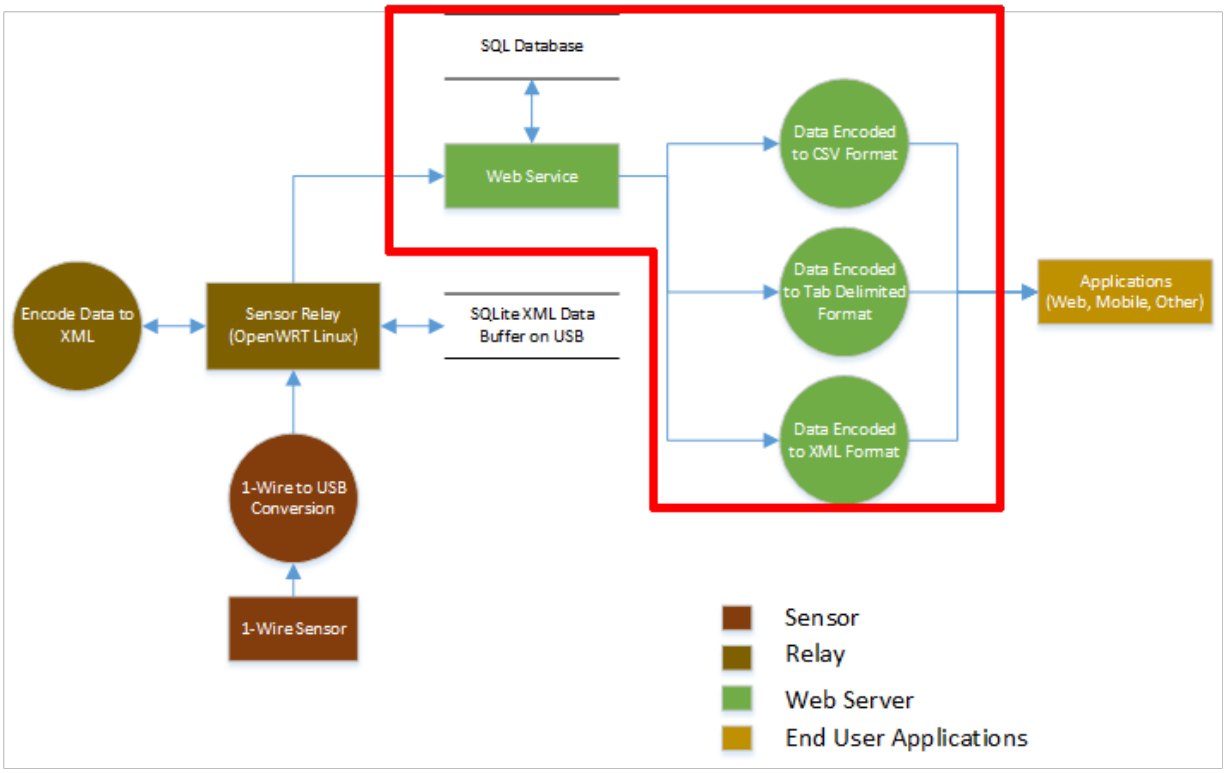


BigSense

BigSense is a RESTful web service, written in Scala, that is designed to handle the collection and retrial of sensor data. It was designed to work with LtSense, but can be extended to work with other sources of sensor inputs. When used with LtSense, it consumes data in the SenseData XML Format and has a Webservice API that can be used to query that data.

Queries can be restricted by timestamp range, date range, sensor number, relay number or combinations of the above constraints. BigSense can also aggregate sensor data, such as given the average temperatures over a given period or showing the average flow rates over a given period. It can handle image data and provides a means for security using RSA certificate verification. Finally, it supports outputs in a number of formats including XML, comma separated values, tab delimited values or HTML tables.

Multiple relational databases are supported including Microsoft SQL Server, MySQL and PostgreSQL (with PostGIS extensions). Its components architecture includes Actions, Conversion, Database Handlers, Formats, Models, Processors, Security Handlers and Validators. Each of these components is fairly independent. They are coupled together using Spring Dependency Injection.



Components

REST and Components

BigSense is a RESTful web service. To this end, the individual components that get called can often been seen in the URL itself. Take for example the following request:

http://example.org/Aggregate/Sum/DateRange/20120501/20120502/60.txt?Timezone=America/New_York&Units=Standard

From this URL we can see the **AggergateAction** class is called. All the other path elements of the URL such as Sum, DateRange, 20120501, 20120502 and 60 are placed in an **ActionRequest** object and sent to that action. Everything after the first period indicates the **FormateTrait** to be used. In this case it's the tab delimited format (txt). Then, two additional **ConverterTrait**s are applied: **TimezoneConverter** and **UnitsConverter**. There are other components that are used behind the scene such as **ValidatorTrait**s and the **DatabaseHandler.**

Actions

Actions use the naming format <Action Name>Action for their class names. The name doesn't actually matter as they are glued into the framework using Spring as we'll see later. The actions do not preform any validation on their inputs. Those are taken care of by **ValidationTrait**s. They simply assume the arguments are correctly formatted and of the right type and do what the call requires. This typically involves either reading or writing data to the database. The inputs to an actions runAction() method is an **ActionRequest** object and the action returns an **ActionResponse**.

Validators

Validatros are called before the appropriate action. If two actions have the same input requirements, they can share the same validator. If the validator returns something other than the Scala **None** type, the error message is processed and sent to the end user without the action being run. If the validation passes, the action is then called.

Formats

TODO paceholder

Model

TODO placeholder

Security

TODO placeholder

Database

Supported Databases

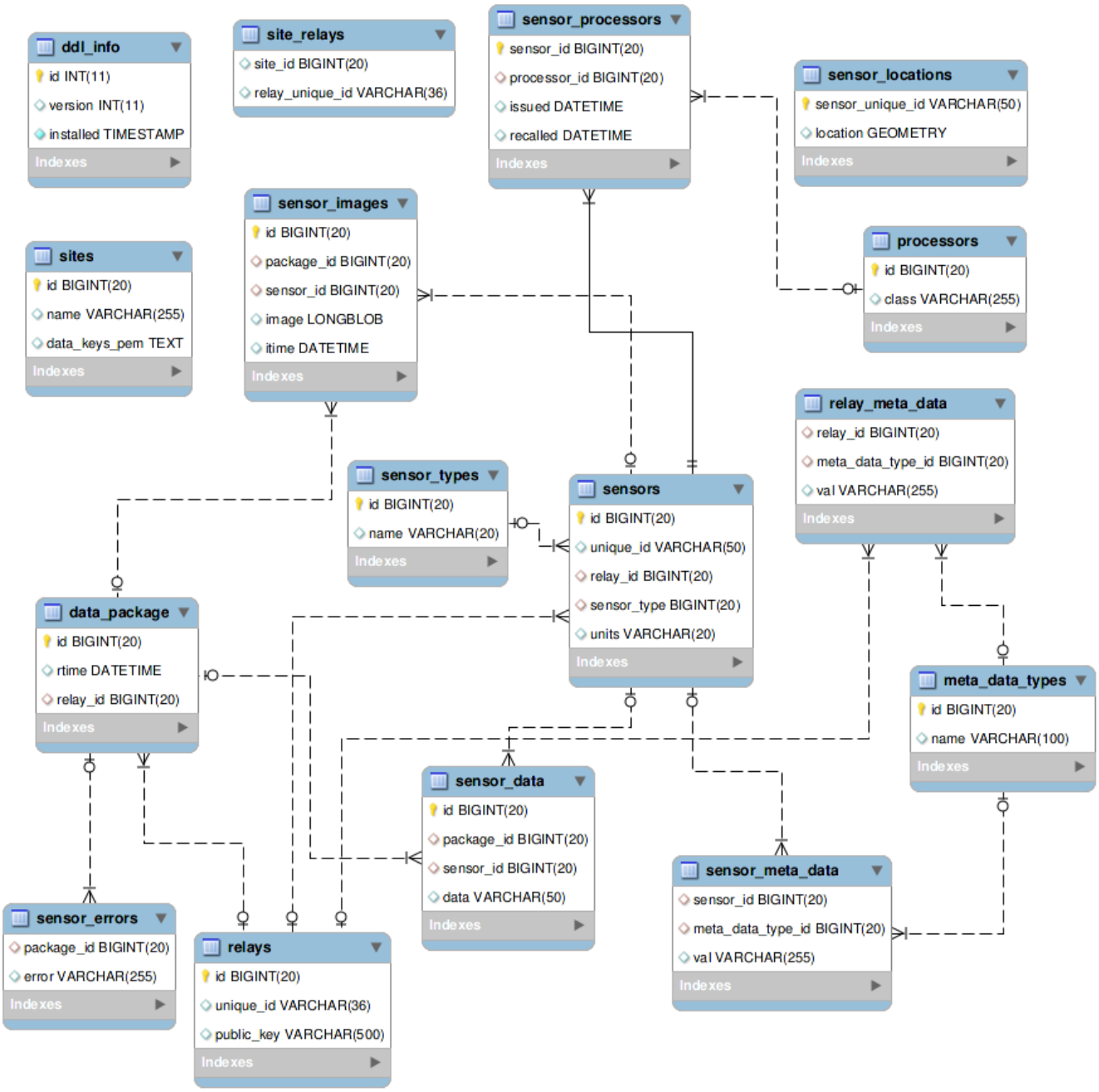
- Microsoft SQL Server
- PostgreSQL (w/ PostGIS)
- MySQL (in-progress)

Development

BigSense does not use a Database Abstraction Object (DAO) layer. Instead, each database backend has its own custom data definition files and a command file for SQL queries.

> **SQL Queries**: https://github.com/sumdog/BigSense/tree/master/src/io/bigsense/db
>
> **DDL**: https://github.com/sumdog/BigSense/tree/master/ddl

Entity Relationship Diagram*



*MySQL Version. Data types may vary for other backends.

Dependency Injection

This section is out of date. BigSense has moved off of Spring for Dependency Injection and AOP. It now has Dependency Injection built into standard Scala code and AOP via AspectJ at compile time. This section will be updated soon to reflect these changes. For now, the source code for the Dependency Injection can be found here:

https://github.com/BigSense/BigSense/blob/master/src/main/scala/io/bigsense/spring/MySpring.scala

Spring Dependency Injection is used to tie all the individual components together. To learn more about Spring Dependency Injection, please read the chapter on Dependency Injection in *Spring in Action* published by Manning. In our Application Context, the *name* and *id* fields are identical for every bean. The following examples are taken from the Application Context in the current version found in BigSense at the time of this writing. They may not be up to date. Please view the latest version at the following URL:

https://github.com/sumdog/BigSense/blob/master/src/io/bigsense/spring/spring.xml

Actions

```
...
   <!-- Actions -->

    <bean id="BaseAction" name="BaseAction" class="io.bigsense.action.ActionTrait" abstract="true">
        <property name="dbHandler" ref="databaseHandler" />
    </bean>

    <bean id="ActionSensor" name="ActionSensor" class="io.bigsense.action.SensorAction" parent="BaseAction">
        <property name="validator" ref="SensorActionValidator" />
    </bean>
    <bean id="ActionQuery" name="ActionQuery" class="io.bigsense.action.QueryAction" parent="BaseAction">
        <property name="validator" ref="QueryActionValidator" />
    </bean>
...
```

Actions are called based on their *id*. The very first argument to the service is post-fixed with the word **Action**, so it is impossible for the user to inject anything and call a bean that isn't explicitly defined as an Action within Spring. Each Action must have a **ValidationTrait** implementation. This is not optional. If an action requires no validation, a blank validation trait must be made to pass all arguments unfiltered to the action.

```
...
    <!--  Validators -->

    <bean id="QueryActionValidator" name="QueryActionValidator" class="io.bigsense.validation.
QueryActionValidator" />
    <bean id="AggregateActionValidator" name="AggregateActionValidator" class="io.bigsense.validation.
AggregateActionValidator" />
    <bean id="SensorActionValidator" name="SensorActionValidator" class="io.bigsense.validation.
SensorActionValidator" />
    <bean id="StatusActionValidator" name="StatusActionValidator" class="io.bigsense.validation.
StatusActionValidator"/>
    <bean id="ImageActionValidator" name="ImageActionValidator" class="io.bigsense.validation.
ImageActionValidator"/>
...
```

Validators are fairly straight forward. They have no dependencies. The base **ValidatorTrait** contains several helper functions to assist in validating arguments.

TODO: Finish this section

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans
/spring-beans-3.0.xsd
        http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.0.
```

```
xsd">

    <!-- Actions -->

    <bean id="BaseAction" name="BaseAction" class="io.bigsense.action.ActionTrait" abstract="true">
        <property name="dbHandler" ref="databaseHandler" />
    </bean>

    <bean id="ActionSensor" name="ActionSensor" class="io.bigsense.action.SensorAction" parent="BaseAction">
        <property name="validator" ref="SensorActionValidator" />
    </bean>
    <bean id="ActionQuery" name="ActionQuery" class="io.bigsense.action.QueryAction" parent="BaseAction">
        <property name="validator" ref="QueryActionValidator" />
    </bean>

    <bean id="ActionStatus" name="ActionStatus" class="io.bigsense.action.StatusAction" parent="BaseAction">
        <property name="validator" ref="StatusActionValidator" />
    </bean>

    <bean id="ActionAggregate" name="ActionAggregate" class="io.bigsense.action.AggregateAction" parent="
BaseAction">
        <property name="validator" ref="AggregateActionValidator" />
    </bean>

    <bean id="ActionImage" name="ActionImage" class="io.bigsense.action.ImageAction" parent="BaseAction">
        <property name="validator" ref="ImageActionValidator" />
    </bean>

    <bean id="ActionReport" name="ActionReport" class="io.bigsense.action.ReportAction" parent="BaseAction">
        <property name="validator" ref="StatusActionValidator" />
        <property name="converters" ref="convertersAsScalaMap" />
    </bean>

    <!-- Security -->
    <bean id="SecurityManager" name="SecurityManager" class="io.bigsense.security.SignatureSecurityManageer" />

    <!-- Processors -->
    <bean id="BaseProcessor" name ="BaseProcessor" class="io.bigsense.processor.ProcessorTrait" abstract="true">
        <property name="dbHandler" ref="databaseHandler" />
    </bean>

    <bean id="CounterProcessor" name="ProcessorCounter" class="io.bigsense.processor.CounterProcessor" parent="
BaseProcessor" />
    <bean id="ImageProcessor" name="ProcessorImage" class="io.bigsense.processor.ImageProcessor" parent="
BaseProcessor" />


    <!-- Smart Vision Analyzers -->
    <bean id="AnalyzerWaterLevel" name="AnalyzerWaterLevel" class="io.bigsense.smartvision.WaterLevelAnalyzer"
/>

    <!--  Validators -->

    <bean id="QueryActionValidator" name="QueryActionValidator" class="io.bigsense.validation.
QueryActionValidator" />
    <bean id="AggregateActionValidator" name="AggregateActionValidator" class="io.bigsense.validation.
AggregateActionValidator" />
    <bean id="SensorActionValidator" name="SensorActionValidator" class="io.bigsense.validation.
SensorActionValidator" />
    <bean id="StatusActionValidator" name="StatusActionValidator" class="io.bigsense.validation.
StatusActionValidator"/>
    <bean id="ImageActionValidator" name="ImageActionValidator" class="io.bigsense.validation.
ImageActionValidator"/>

    <!--  Convertors -->
    <bean id="convertersAsScalaMap" class="scala.collection.JavaConversions" factory-method="mapAsScalaMap">
        <constructor-arg ref="converters"/>
    </bean>

    <util:map id="converters">
          <entry key="Units">
```

```
            <bean class="io.bigsense.conversion.UnitsConverter" />
        </entry>
        <entry key="Timezone">
            <bean class="io.bigsense.conversion.TimezoneConverter" />
        </entry>
    </util:map>


    <!-- Formats -->

    <bean id="FormatAGRA.XML" class="io.bigsense.format.SenseDataXMLFormat" />
    <bean id="FormatTXT" class="io.bigsense.format.TabDelimitedFormat" />
    <bean id="FormatCSV" class="io.bigsense.format.CSVFormat" />
    <bean id="FormatFLAT.XML" class="io.bigsense.format.FlatXMLFormat" />
    <bean id="FormatTABLE.HTML" class="io.bigsense.format.TableHTMLFormat" />


    <!-- logging  -->
    <bean name="logger" class="io.bigsense.spring.LoggingInterceptor" />
    <bean id="loggingAdvisor" class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
      <property name="advice" ref="logger" />
      <property name="pattern" value="io.bigsense.*" />
    </bean>
    <bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator" />


    <!-- Database Handler -->
    <bean id="databaseHandler" class="io.bigsense.db.DatabaseHandler">
        <property name="ds" ref="mainDataSource" />
        <property name="converters" ref="convertersAsScalaMap" />
        <property name="sqlCommands" ref="sqlMicrosoftCommands" />
    </bean>


    <bean id="sqlMicrosoftCommands" class="io.bigsense.util.SQLProperties">
        <constructor-arg name="sqlCommandFile" value="/io/bigsense/db/mssql.commands"/>
    </bean>


    <!-- Database configuration -->
    <bean id="mainDataSource" class="com.jolbox.bonecp.BoneCPDataSource" destroy-method="close" >
       <property name="driverClass" value="net.sourceforge.jtds.jdbc.Driver" />
       <property name="jdbcUrl" value="connectionString" />
       <property name="username" value="dbUser"/>
       <property name="password" value="dbPass"/>
       <property name="idleConnectionTestPeriod" value="60"/>
       <property name="idleMaxAge" value="240"/>
       <property name="maxConnectionsPerPartition" value="30"/>
       <property name="minConnectionsPerPartition" value="10"/>
       <property name="partitionCount" value="3"/>
       <property name="acquireIncrement" value="5"/>
       <property name="statementsCacheSize" value="100"/>
       <property name="releaseHelperThreads" value="3"/>
    </bean>

</beans>
```

SenseData XML Format

Example

```
<?xml version="1.0" ?>
<sensedata>
    <package id="Sumit-Local-Laptop" timestamp="1346113433203">
        <gps>
           <location longitude="38.446" latitude="96.334" altitude="100.34" />
           <delta climb="0.0" track="130.4" speed="0.0" />
           <accuracy longitude_error="1.0" latitude_error="2.5" altitude_error="" climb_error="" track_error=""
speed_error="" />
        </gps>
        <sensors>
            <sensor id="SUMTEMP001" type="Temperature" units="C">
                <data>11</data>
            </sensor>
            <sensor id="SUMVOL001" type="Volume" units="l">
                <data>484</data>
            </sensor>
            <sensor id="SUMFLO001" type="FlowRate" units="l">
                <data>776.300</data>
            </sensor>
        </sensors>
    </package>
</sensedata>
```

SenseData elements can contain multiple package elements. Each package contains the Relay ID listed as just id and a timestamp which is a standard UNIX timestamp in seconds. Each package should contain only one sensors element. The sensors element can have multiple sensor elements, each with an id (Sensor ID), type and units. The sensor can contain one data element with the sensor data in its TEXT field.

Valid Sensor Types

- Temperature
- Volume*
- FlowRate*

*Volume and FlowRate are typically both components of the same counter. For 1-Wire devices in LtSense, currently a single counter will be represented by two sensor elements, one where the id ends with a -FR and the other with a -V for FlowRate and Volume respectively.

**Capitalization matters.

Spatial Data

Location data is optional and can be specified using the gps tag. All three sections within the gps tag, location, delta, and accuracy are optional. However if a tag is present, **all attributes within that tag are required.** If data is not available for a given field, simply use the empty string ("").

- location - Contains a longitutde, latitude and altitude. Both longitude and latitude must be present together or absent (if you only have an altitude sensor, you can only report the altitude, but the other two fields must be present and blank). Longitude or Latitude by itself without the other is not valid.
- delta - Contains fields relating to rates of change. speed is speed in meters per second, track is course over ground (degrees from true north) and climb is vertical ascent/descent speed in meters per second
- accuracy - Potential +/- error for each field longitude_error, latitude_error, altitude_error, speed_error, climb_error, and track_error

Future Sensors

- pH
- Humidity
- Soil Moisture

Security

If security is enabled within BigSense, then a signature will need to be attached to the XML data. The attachment should be a Base64 encoded RSA with SHA1 digest signature of the XML data in its transmitted form, not including any leading or trailing whitespace or newlines. The signature must be separated by two newline characters. Both XML data and the signature should be in the HTTP POST Body. The following is a full POST request from a client to the BigSense web service from the LtSense client:

```
POST /bigsense/api/Sensor.sense.xml HTTP/1.1
Accept-Encoding: identity
Content-Length: 703
Host: localhost:8080
Content-Type: application/x-www-form-urlencoded
Connection: close
User-Agent: Python-urllib/2.7

<?xml version="1.0" ?><SenseData><package id="Sumit-Local-Laptop" timestamp="1346251838355"><sensors><sensor
id="SUMTEMP001" type="Temperature" units="C"><data>1</data></sensor><sensor id="SUMVOL001" type="Volume" units="
l"><data>488</data></sensor><sensor id="SUMFLO001" type="FlowRate" units="l"><data>776.659</data></sensor><
/sensors></package></SenseData>

uWKUWtrqkL7B81K7jV47k2Kw7neOdl3npxao4zCk4tVgcdMeaj7IDMrVnw0wKYcsX7YMftHg83GRsf+Ovtu8ZC6OymgdpR4mAzb3jdk7cOBoDXLq
0/O88TVLASHzhZ4BPu4qv0K8jQ/omSix1T49zYUeKL8jTHD+AOXBzksGX2J6SHkWLvTbUUqwkgMOwcm6sMO7Xb/f+aq
/UFRkpafvHjHfCzLXwD3RGte4qqGgM+K+bpJOmv4OGZyx+FozXo5FAqYn7O/OA1JkzGnyPrUXxVdUKcKg+njDIcTl3CNeuUt6ZNWhxIO
/LPtD9j0uI2HLH/b4mpNSPzcbAGeMOBe4Kg==
```

For information on how to build these signatures in both Python 2 and Java, please refer to http://penguindreams.org/blog/signature-verification-between-java-and-python/.

Webservice API

BigSense Web Service REST API

Function Matrix

Methods	Action	Arguments	Constraints	Description	Example
GET POST	Sensor	package_id (GET only)		Read/Write Sensor Packages	POST /Sensor.sense.xml GET /Sensor/15.sense.xml
GET	Query/Latest	limit (required)	SensorID RelayID SensorType WithinMetersFrom	Query Latest Sensor Readings	GET /Query/Latest/100.csv
GET	Query /TimestampRange	start/end (UNIX time in seconds)	SensorID RelayID SensorType WithinMetersFrom	Query Readings between timestamp range	GET /Query/TimestampRange /1318711900/1319119800.txt
GET	Query /DateRange	start/end (YYYYMMDD - ISO Date Format)	SensorID RelayID SensorType WithinMetersFrom	Query Readings in date range	GET /Query/DateRange /20111001/20111005.csv
GET	Query/Relays			List of Relays	GET /Query/Relays.txt
GET	Query /Sensors			List of Sensors	GET /Query/Sensors.csv
GET	Aggregate /Average /DateRange	start/end/interval (YYYYMMDD - ISO Date Format, interval in minutes)	SensorID RelayID SensorType	Shows the mean average of numerical data for each sensor subdivided by interval from a given date range	GET /Aggregate/Average /DateRange/20111225/20121230 /60.table.html
GET	Aggregate /Average /TimestampRange	start/end/interval (UNIX time in seconds, interval in minutes)	SensorID RelayID SensorType	Shows the mean average of numerical data for each sensor subdivided by interval from a given timestamp range	GET Aggregate/SumVolume /TimestampRange/1318711900 /1319119800/60.table.html
GET	Aggregate /Sum /DateRange	start/end/interval (YYYYMMDD - ISO Date Format, interval in minutes)	SensorID RelayID SensorType	Shows the sum of numerical data of each sensor subdivided by interval from a given timestamp range	GET /Aggregate/AvgTemp /DateRange/20111225/20121230 /60.table.html
GET	Aggregate /Sum /TimestampRange	start/end/interval (UNIX time in seconds, interval in minutes)	SensorID RelayID SensorType	Shows the sum of numerical data of each sensor subdivided by interval from a given date range	GET Aggregate/AvgTemp /TimestampRange/1318711900 /1319119800/60.table.html

Formats

- SenseData XML (sense.xml): Only used in transmitting data from the client to the server. Includes timezone, sensor information and can contain multiple packets
- TABLE.HTML: A human readable, sortable HTML table that can be viewed in a web browser
- TXT: Tab Delimited flat data format
- CSV: Coma Separated flat data format

Action	agra.xml	txt	csv	table.html
Sensor	SUPPORTED	SUPPORTED	SUPPORTED	SUPPORTED
Query	UNSUPPORTED	SUPPORTED	SUPPORTED	SUPPORTED
Aggregate	UNSUPPORTED	SUPPORTED	SUPPORTED	SUPPORTED

Constraints

Some query functions support constraints passed at GET parameters

- **SensorID:** Restricts results by Sensor Unique Identifier
- **RelayID**: Restricts results by Relay Unique Identifier
- **SensorType:** Restricts results by Sensor Type

- **WithinMetersFrom**: Restricts results to within a radius (in meters) of a specific longitude, latitude.
    - The following example shows a restriction from within 1 kilometre of longitude 34.6 and latitude 96.543
    - http://localhost/Query/Latest/100.txt?WithinMetersFrom=34.6long96.543lat1000r

Example:

http://localhost/Latest/200.txt?SensorType=Temperature&RelayID=4-Paver1

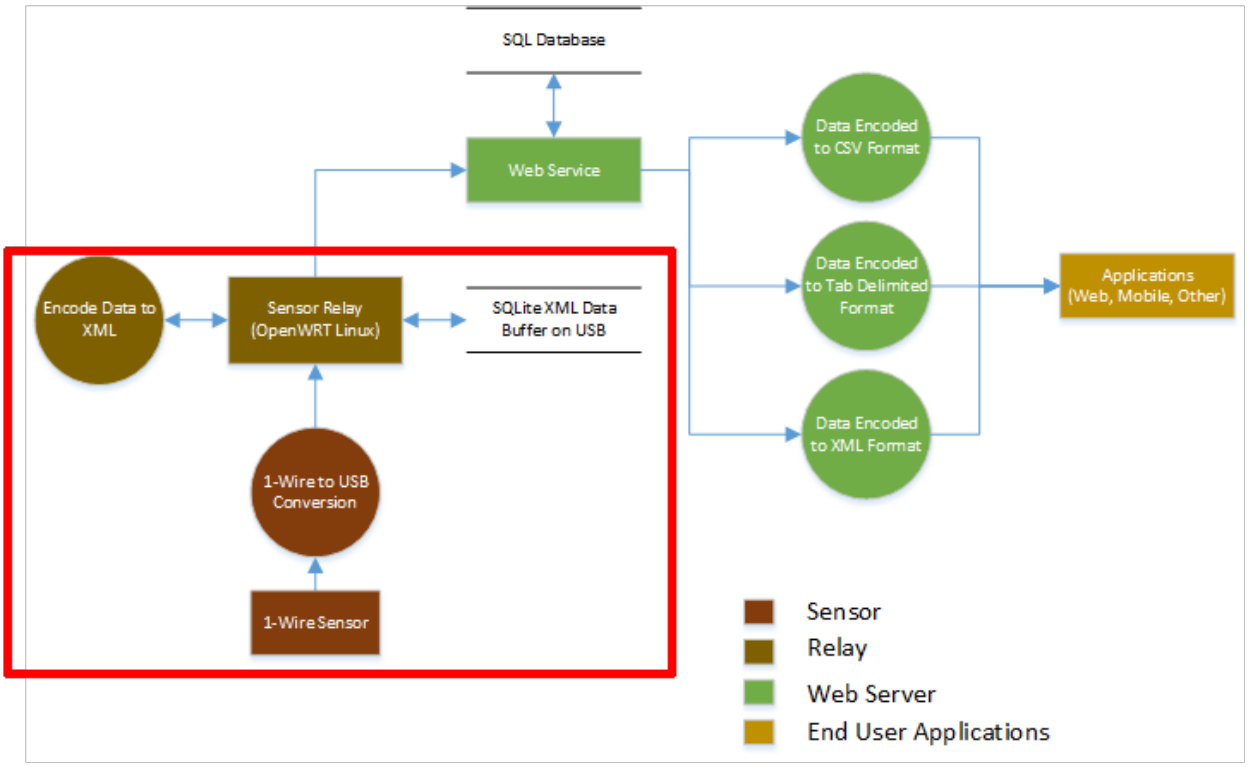
Converters

Query functions also support conversions that are passed as GET parameters similar to constraints.

Converter	Valid Arguments	Description
Units	Metric/Standard	Converts units between metric and standard (Default: metric)
Timezone	EST5EDT, CST, UTC, etc.	Converts time fields to other time

LtSense

LtSense is the Python2 client designed to run on embedded Linux systems (such as OpenWRT) and communicate with the BigSense web service. It supports polling sensors at a set interval and queueing the data for reliable transmission to a destination web service. The base components for LtSense can easily be extended as seen in the Configuration section for LtSense.



Sensors

Valid Sensor Types

- Temperature
    - DS18B20 1-wire
- Volume*
- FlowRate*
- Humidity
- Light
- Sound

*Volume and FlowRate are typically both components of the same counter. For 1-Wire devices in LtSense, currently a single counter will be represented by two sensor elements, one where the id ends with a -FR and the other with a -V for FlowRate and Volume respectively.

*All 1-wire within OWFS are supported

**Capitalization matters.