

Technote

AirVantage™ Development Tools for Quick Data Retrieval from General Electric Meters



Graeme Harfman & Mike Aksmanovic
Sierra Wireless

TECHNOTE:

Using AirVantage™ Development Tools to Easily Retrieve Data from ANSI 12.19 Electric Meters

Problem Definition

The electrical utility industry is challenged with significant energy consumption growth and an aging infrastructure. The development of a Smart Grid enables remote data collection and dissemination required to support end-user demand response programs and multi-source energy generation, as well as overall supply obligations. A Smart Grid requires the utility infrastructure, such as meters, transformers, and protection and control equipment, to transfer data to a central server in order for the utility to use the information to improve the efficiency of power generation and distribution. And acquiring data from electrical meters dispersed across the country requires a wireless module to interface to the meter.

The AirVantage™ Platform is an M2M wireless data collection and operational support platform used to collect data from the utility infrastructure. AirVantage offers development tools that allow the developer to:

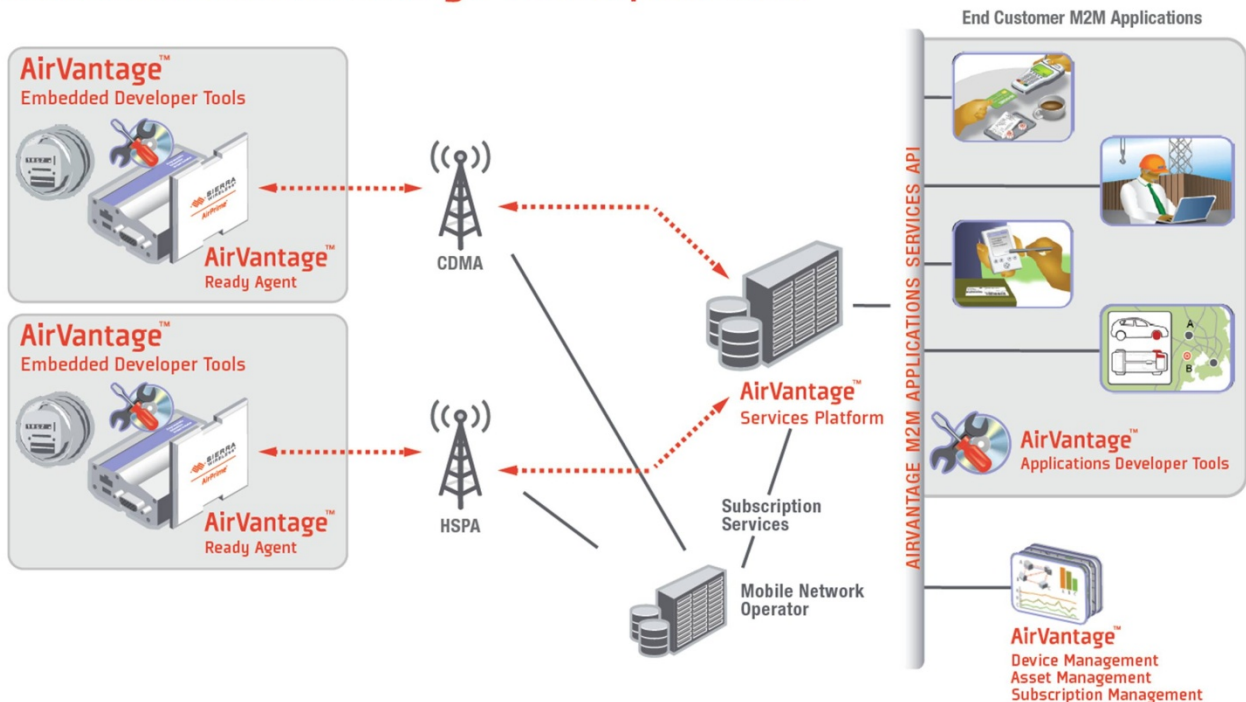
- Quickly connect to the meter interface
- Build a complete infrastructure to transfer data to a central server
- Present data as information
- Integrate data into utility infrastructure

This technote will show how a developer can use the AirVantage Developer Suite to create an interface to a GE kv2C Electrical Meter quickly and display that information on a web page.

Development Tools Used

AirVantage Developer Suite is an Eclipse-based toolset (<http://www.eclipse.org>) that consists of embedded and enterprise application development tools. The suite is extensible with the Eclipse OSGI plug-in model and supports many languages and frameworks, including embedded C, Java, and Lua scripting, as well as enterprise application development in .NET, PHP, or Java. The suite is compatible with standard Eclipse-based integrated development environments (IDEs), such as IBM Websphere, SpringSource Tool Suite, Oracle Enterprise Pack for Eclipse, and the Eclipse Web Tools Platform.

Sierra Wireless AirVantage™ Developers Suite



Embedded Development Tools

The AirVantage Developer Suite embedded developer tools leverage the development and infrastructure capabilities of AirVantage Ready Agent to support embedded C, Lua, and Java integration to asset data - in this case, integration to the meter's interfaces. The tools also support a graphical component-modeling tool based on the Eclipse Modeling Framework. The component is compatible with existing embedded software IDEs, including Sierra Wireless Developer Studio for OpenAT development, Wind River Workbench, TI Code Composer, and J2ME.

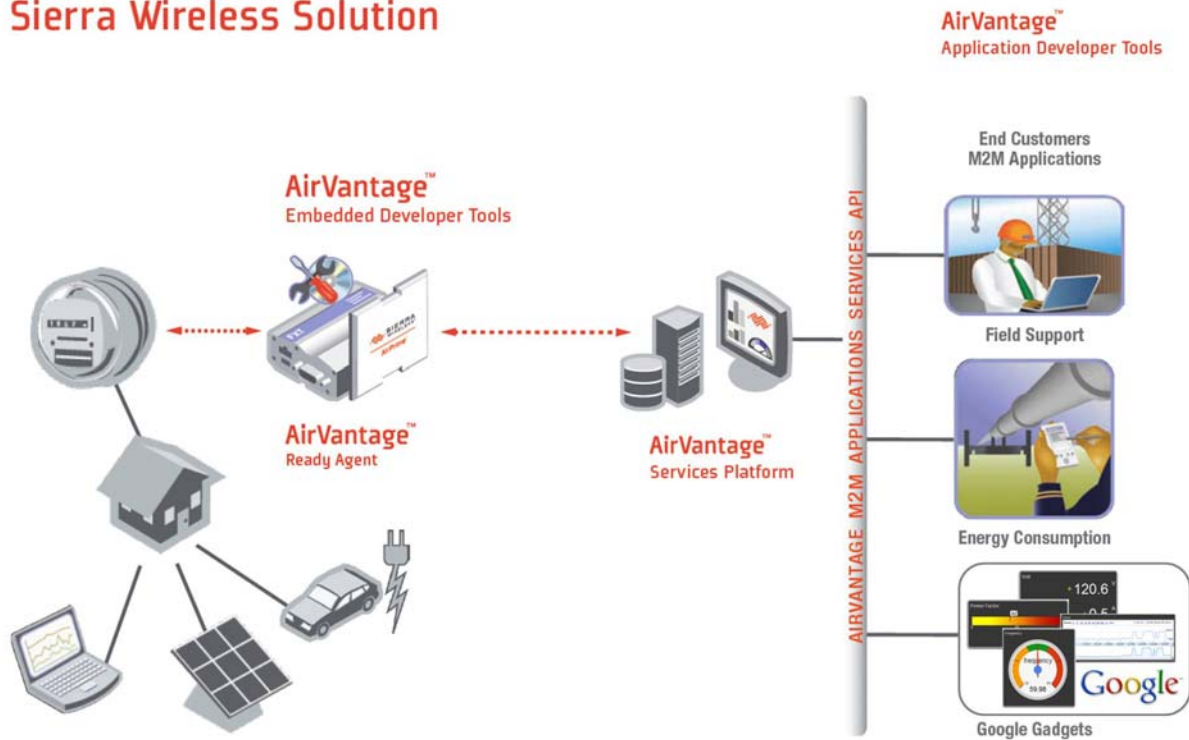
Application Development Tools

The AirVantage Developer Suite application developer tools leverage the AirVantage Platform M2M Application Services API and the underlying AirVantage infrastructure. The M2M Application Services API comes with .NET, PHP, and Java integration examples and is compatible with many third-party Web Services application development tools. In this example, we will connect the data output from the GE meter to Google Gadgets to demonstrate the meter's dynamic data presented on a web page.

The Solution

The RS232 output from a GE meter is connected to a RS232 input of an AirLink GL6100 device. The AirLink® gateway hosts the Ready Agent, which communicates over a mobile broadband wireless network to the AirVantage Platform.

Sierra Wireless Solution



The first step was to understand how to get information out of the meter. The GE kv2C Electrical meter uses the ANSI 12.19 protocol. The process for integrating a meter supporting the IEC 62056 protocol is similar. In this example, we extract some simple values, including instantaneous power consumption, voltage, current, frequency and power factor, that can change immediately with changes to system load. The asset (meter) model that describes the electrical meter data elements is defined. Only one asset model in the developer suite needs to be defined, and this defines the data elements for both the

embedded and the server development. This way, both the server and the embedded programs know what to send and receive, and the data dictionaries are automatically synchronized.

On the embedded side, now that the protocol and the data we want is understood, a new embedded development project is started to interrogate the meter. It was decided to use the modeling framework to begin development of the software. The development was broken down into three components. The modeler was first used to set up the wireless connection, set up access to the serial port, create a timer to periodically read the meter, and send data to the portal.

In the figure below, the boot-up sequence of the device is shown. It sets up the wireless connection, synchronizes the time, sets up the serial port (UART), and configures the timer.

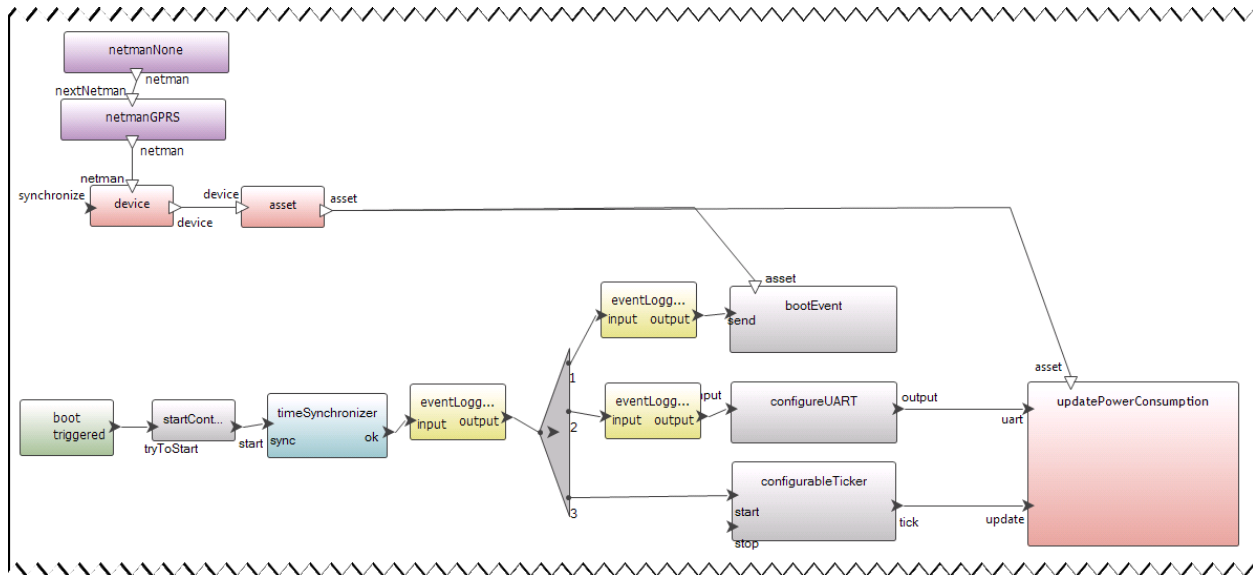


Figure 1: Wireless, serial, and ticker initialization

Each of the pre-built elements has configurable settings that give the developer control over how to set up each element. For example, the ‘configurableTicker’ element has a time setting that allows the developer to decide when the timer should trigger the meter read.

To read information from the meter, a subset of the ANSI 12.19 protocol was implemented. Lua scripting was used to take advantage of the methods that the ‘configureUART’ element in Figure 1 made available. In Figure 2, the start of the login process to the meter is shown. This is done by sending a sequence to the meter and then waiting for the response in the serial port.

```

self :ge() :write ( { 0xEE, 0x00, 0x00, 0x00, 0x00, 0x01, 0x20, 0x13, 0x10 } )
local emitter, event, x = wait ('e', 'frame')
  
```

Figure 2: Meter login

The meter responses have a check sum calculated to ensure that the data is not corrupted. The CRC code was already available as a C function called “parsecrc.” This needed to be integrated into the Lua code. Making the C function available from the Lua scripting is shown in Figure 3.

```
void luaopen_parseCRC(lua_State *L) {
    lua_pushcfunction(L, parsecrc);
    lua_setglobal(L, "parsecrc");
}
```

Figure 3: Include C code

Figure 4 shows part of the C CRC code that was used:

```
static int parsecrc(lua_State *L) {
    const char *s = luaL_checklstring(L, 1, NULL);
    int i = (int)((unsigned char)s[0]) - 2;
    unsigned short crc_value;

    crc_value = crc (i, (unsigned char *)&s[1]);

    if (crc_value == (256 * (unsigned char)s[i+1] + (unsigned char)s[i+2])) {
        i = 1;
        lua_pushinteger(L,i);
    } else {
        i = 0;
        lua_pushinteger(L,i);
    }
    return 1;
}
```

Figure 4: C CRC code

The CRC allowed the validity of the response from the meter to be checked. If everything checked out, the program continued interacting with the meter until all the required information was extracted from the meter.

The Lua code stored the results into a Lua list that could then be sent to the portal. The subsection of the model used to send the data items to the portal is shown below in Figure 5. The configuration of the data elements is based on the asset model that was defined above.

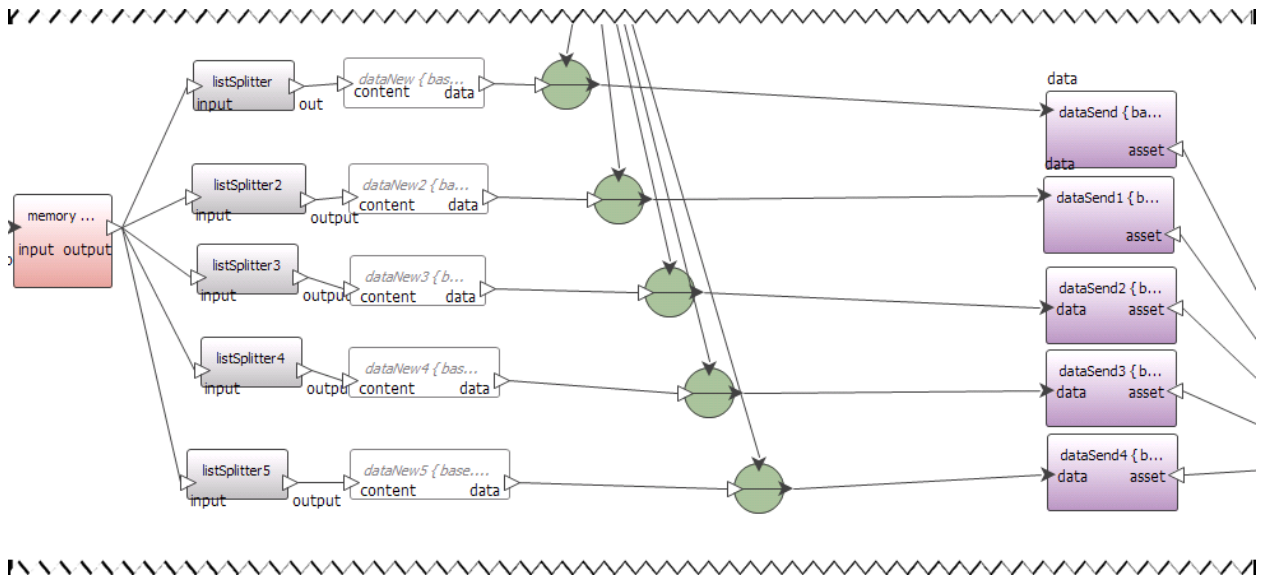


Figure 5: Sending data to the portal

Finally, the data in the portal needs to be shown on a web interface. The Google Gadgets that are included in the developer suite were used to show the data. The gadgets are able to take advantage of the asset model that was already created to configure how each data element must be shown. From the GE meter project, several selections are possible, including the ability to create a new Android application, a Java Spring MVC controller, or an Open Social Gadget.

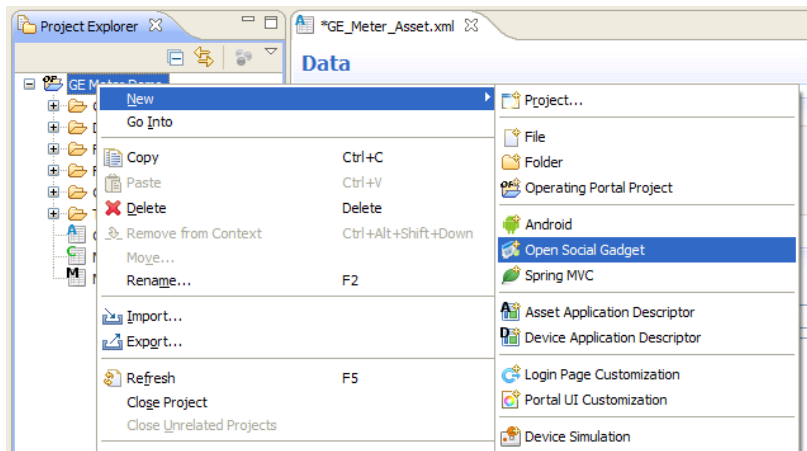


Figure 6: Create new Open Social Gadget

The toolkit provides a number of pre-built gadgets. In this case, the one that displays two values is selected.

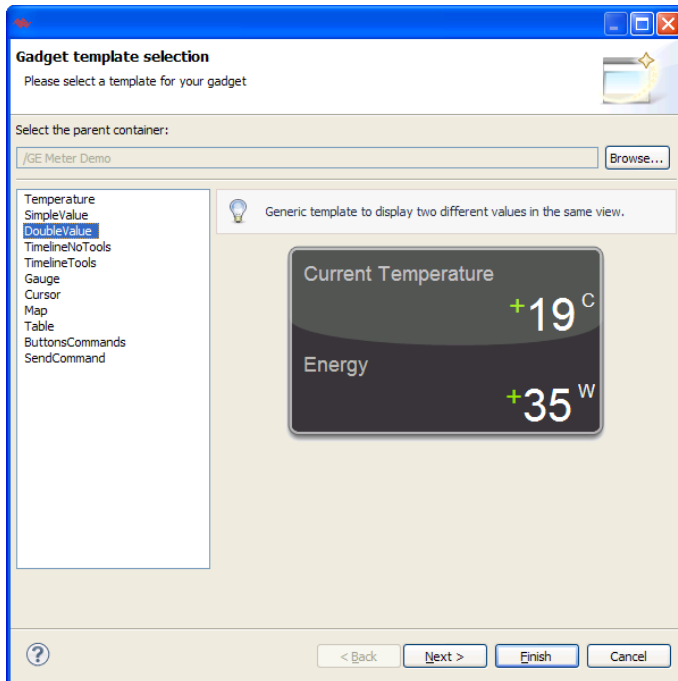


Figure 7: Gadget selection

When filling out the values, the variables that are presented are the ones defined in the asset model, and these can quickly be added to the gadget.

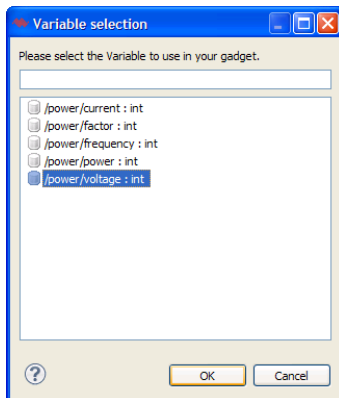


Figure 8: Available variables

Finally, they are deployed to an iGoogle page, with some sample gadgets and values shown below:



Figure 9: Sample report page

This data can easily be integrated into other applications using the available M2M Application Services API, including billing and SCADA systems.

Results

The development of this project went very well after the initial understanding of the serial port format. Embedded developer tools supporting C, Lua, and the graphical modeler tools were used to connect the meter to an AirLink gateway and to send data to the AirVantage Platform. The AirVantage Platform was used to host the data and to present the data to AirVantage M2M Application Services API. The Application development tools quickly connected the AirVantage M2M Application Services API to a set of Open Social Gadgets on an iGoogle page. The infrastructure provided with AirVantage was leveraged to complete this project in about 16 hours. The Ready Agent pre-integrated into AirLink hardware and all of its development and messaging services to the AirVantage Platform, which saved significant development and debugging effort. The common data model replicated on the device and the server, and then available to the Application Services API, also saved significant database development and support.

