For Those who Would like to Understand More about SDD, but Were Afraid to Ask

Version 1.4 November 17, 1997

Mark A. Whiting
Smart Trek MMDI
1.509.375.2237
Mark.A.Whiting@pnl.gov

The Smart Trek SDD Development team is led by Dr. Daniel Dailey at the University of Washington
See: http://www.its.washington.edu/bbone/ for more information.
Self-Describing Data for Dummies

Introduction

Self-Describing Data (SDD) is an approach to transmitting and delivering data, such as transportation data, where a stream of actual data is prefixed with a set of data that “describes” it. This descriptive data can be anything. Yet the intent is to provide descriptive data that helps users, and applications, to understand the actual data that follows. An example data stream is the traffic loop sensor data collected by the Washington State Department of Transportation (WSDOT), where over two thousand sensors provide new data every twenty seconds. The descriptive data for this stream consists of descriptions of the sensors, their cabinets, their locations, and so on. This preface plus the continuous actual data stream create an SDD stream.

SDD is often difficult to understand as it is a new approach to data packaging and transmission and because the approach is still evolving. This document attempts to provide an introduction to SDD for those of us trying to keep up.

What is Self-Describing Data?

Self-Describing Data (SDD) is a data stream that consists of descriptive data, followed by a continuous stream of data. Figuratively, an SDD stream would appear as:

![Diagram](image)

In terms of traffic data, this stream might appear (using English prose):

“You’ll be getting traffic loop sensor data; that belongs to these cabinets X, Y, Z; from locations A, B, and C. The format is “loop sensor name”: 8 bytes, “occupancy”: 16 bytes, and “volume”: 8 bytes.”

“XXX-4583” 50 3
“XXX-4587” 43 2
“XXX-3848” 48 2
...

More on the SDD format shortly...

What Kind of Actual Data Should be Made Self-Describing?
Any kind of data can be delivered as self-describing. But if you have an unchanging (static) set of data, such as a set of reports, or a very slowly changing set of data, such as the King County Metro Transit schedule data that is updated only three times a year, it may not be worth encoding and delivering this data using the SDD approach.

It may be that the best kind of data to be delivered in SDD format is real-time (or near-real time) data streams such as the WSDOT traffic loop sensor data or the King County Metro Transit bus location data. The data elements of these streams are constantly changing in content (remember loop sensor data is updated every 20 seconds), and they have a rich set of descriptive data that can be used by applications in providing traffic management information, traveler information, and so on.

**About the Description**

The real-time or actual data may be very simple in structure. Even if the actual data is simple, the descriptive data component of an SDD stream may be quite extensive and complex. This complexity is a confusing aspect of SDD. Elements of a simple data stream may have a lot of hidden or implicit meaning (We’ll see an example of this below.). If users do not know this, data can be misinterpreted or misused. SDD tries to make more of this implicit meaning available through the descriptive data. Because of this potential complexity, descriptive data is encoded as a database, using database structures and language. This was designed so that applications could use relational database technology to manage large and complex data descriptions, as opposed to having to derive ad-hoc management schemes. The database is sent in two parts: a specification for the structure (or “schema”) and the contents. So, changing the English prose we used above around a little bit, the stream would more closely look like:

```
“Create one data base table that contains descriptions of the traffic loop sensors. Include a field for loop_id (16 bytes), a cabinet_id (7 bytes), etc. Create another database table that contains descriptions of the loop data. Include a field for sensor_id (15 bytes), volume (int), occupancy (int), etc....”

“Put the following data in the Sensor table:
   “XXX-3848”, “YY1”, ...
   “XXX-4959”, “YY2”, ...
Put the following data in the Loop table:
   ...
   “XXX-4583” 50 3
   “XXX-4587” 43 2
   “XXX-3848” 48 2
   ...
```
This gives us a specification of the data base tables that describes the traffic loop sensors. This is followed by the contents to insert into those tables. Finally, the data stream is sent.

The descriptive data, that the SDD development team calls a data dictionary, is not actually encoded in English, but in a database language called SQL-92. SQL-92 is a standard database language that will allow developers to pass these commands directly to a database and have it understand and perform the appropriate actions (hopefully, create the database and store the values). In SQL-92, our English prose above looks more like:

```
CREATE SCHEMA     <<<this is the structure or schema definition>>>
CREATE TABLE SENSORS
    (LOOP_ID CHAR (16),
     CABINET_ID CHAR (7),
     ...)
CREATE TABLE LOOP_DATA
    (SENSOR_ID CHAR (15),
     VOLUME SMALLINT,
     OCCUPANCY SMALLINT,
     ...)
TABLE SENSORS
    COLUMN (LOOP_ID, CABINET_ID...)   <<<this fills the tables we created >>>
     "XXX-3848", "YY1"...
     "XXX-4959", "YY2"...
TABLE LOOP_DATA
    ...                                  <<<then finally the data...>>>
     "XXX-4583"  50  3
     "XXX-4587"  43  2
     "XXX-3848"  48  2
```

This specification is the same as the previous one, only written using SQL-92 language\(^1\). The CREATE instructions would allow a database to create the specified tables. The TABLE specification with its COLUMN dependents allows the database to fill the tables with the included data. Finally, the data stream trails as before.

---

\(^1\) Actually, the section that describes the contents of the data dictionary is written in a SQL-like language that the SDD development team calls their “Content Language.” The Content Language specifications are translated into SQL-92 by the SDD software.
More Wrapping for SDD

The story of the SDD format doesn’t end with SQL-92. SDD streams are packaged much like Christmas presents with many layers of wrapping paper. To be compliant with a computing standard known as **ASN.1**, the data dictionary and the actual data are wrapped using Basic Encoding Rules (**BER**). Why ASN.1? (Which, by the way, is pronounced “ay-ess-enn-dot-1.”) It’s much too long a story for this paper, but it is a standard the National Architecture and other ITS standards groups prefer. Using BER means give each of the three components of SDD a type, a length and a value. The SDD development team has created software that includes a BER encoder on the SDD transmission side, and a BER decoder on the SDD receiver side. The SDD development team has created software that does this automagically, so we really don’t have to worry about it too much. But if anyone asks if SDD is ASN.1 compliant, we can say “Yes!”

The BER-encoded SDD streams are then also encoded into “ITS Frames.” An ITS Frame is a package developed by the SDD development team to make the network transfer easier. There is an ITS “Framer” at the SDD transmission side and an ITS “Deframer” on the SDD receiver side. The SDD code performs this work automagically as well, so we really don’t have to worry about it either. Below is a diagram of the flow of SDD from a source generating data to a sink that would use it.

![Diagram of the flow of an SDD stream](image)

**Figure 1 The flow of an SDD stream**

What does an SDD stream look like that is BER-encoded and ITS-framed? You really don’t want to see it. It’s best to just think about SDD in terms of the original three components: the Data Dictionary Schema, the Data Dictionary Contents, and the actual Data.

**What goes into a Data Dictionary?**
We gave some examples above as to what might go into a data dictionary. They are not far from what has actually been encoded into one. The SDD development team has created extensive data dictionaries for WSDOT traffic loop sensors and an SDD stream is available with this data for ISPs and other interested parties to use. Let’s look at that dictionary—it has 8 tables.

Here are the tables to be defined as part of the database:

- **COORDINATES** - This table describes coordinate data types, such as “geodetic”; and their measurements, such as “longitude” and “latitude”; and their units of measure, such as “degrees”.

- **MEASURES** - This table provides additional data on coordinate data types, such as the fact that WSDOT uses NAD23 coordinate referencing, while UW uses NAD89.

- **STATION_FLAGS** - This table provides flag values as to whether or not data is usable.

- **INCIDENT_DETECT** - This table provides flag values as to whether an incident has occurred or not.

- **CABINETS** - This table provides cabinet IDs, freeway names, text descriptions, and whether there is a ramp or not.

- **CABINET_LOCATION** - This table provides the location of cabinets. It includes the cabinet ID, the coordinate type used (see above), and whether the data type is defined using the WSDOT or the UW methods, and the location.

- **LOOPS** - This table describes the loop sensors. It contains the loop ID, the cabinet ID, whether or not it’s metered, the road type, the direction of the traffic, the lane type, the lane number, and the sensor type code (a number).

- **ALG_DESCRIPT** - This table provides a complete listing of Java code that will extract the loop sensor data!

As you can see, there is a lot of data here about the data. Heck, there’s even the code to use it. If you didn’t have access to this descriptive data, you would not automatically be aware of information such as locations and sensor types and so on. SDD makes that implicit data explicit and available. The user of SDD is not forced to use any of this descriptive information, but since it’s being provided...why not?
When Do You Get a Data Dictionary and When Do You Just Get Data?

The data dictionary is sent upon connection to an SDD transmitter. An SDD transmission rule is that the data description remains valid until something changes in the data. So the data dictionary is sent once and is not sent again until there is a data change. When a data change does occur, a new data dictionary is sent through the stream. Again, if a data stream were being processed, and a change occurred in the data, the following might appear (going back to the English prose):

```
“XXX-4583” 50 3
“XXX-4587” 43 2
“XXX-3848” 48 2
“XXX-4589” 43 2
“XXX-3850” 48 2
```

“You’ll now be getting traffic loop sensor data; that belong to these cabinets X, Y, Z; from locations A, B, and C. The format is ‘loop sensor name’: 8 bytes, ‘occupancy’: 16 bytes, ‘volume’: 8 bytes, and ‘validity flag’: 1 byte.”

```
“XXX-4583” 50 3 1
“XXX-4587” 43 2 0
“XXX-3848” 48 2 1
```

...  

The change that occurred was that a new data element was added to the stream called “validity flag.” The reason for the new description is to allow users and smart applications that use SDD to handle the data change gracefully and perhaps even adapt to the change and continue functioning normally and without significant loss of service.

How Do You Know Whether or Not You Received a New Data Dictionary?

That is, without doing a lot of searches, data comparisons, and so on. It wasn’t mentioned before, but the SDD stream also includes a time-stamp that is updated when the data dictionary changes. If an application sees a new time stamp, it knows something has changed. And yes, the SDD development team remembered to use four digits to represent the year portion of the time-stamp.

Software for SDD

We’ve mentioned an SDD transmitter and an SDD receiver, in terms almost like a television station transmitter and a home TV receiver, and this is just about how they work. Most ISPs are concerned with the “TV receiver” part of this setup, as they want to get traffic data. They are both pieces of software that projects can use to send and
receive SDD. The receiver is the more mature piece of software at this writing, so we will review this.

The SDD receiver is a Java software program that can receive and understand the Data Dictionary and receive and extract the actual data from the SDD stream. The receiver has some quality checks in it that make sure the SQL-92 language is in the right format and the schema part is in sync with the contents part. The SDD receiver’s primary job is to generate two files:

1. An ASCII file that contains the SQL-92 code that can create the database for all of the descriptive data. This file can be given to a database system, such as ACCESS or SYBASE, to import the data.
2. An ASCII file that contains the actual data.

So what do you do with this, if you’re an ISP? Write software that can dump the Data Dictionary SQL-92 code into a data base, and write other software that can do something interesting with the data stream. That’s the basic stuff. More advanced efforts will work with the newly formed database to provide very cool information to traffic managers or travelers.

**How Do I Tap Into an SDD Data Stream?**

SDD is delivered over the ITS INFORMATION BACKBONE (a.k.a. the “I2B”, pronounced “eye-too-bee”). You plug into it using an Internet (TCP/IP) connection. Programmers know how to do these things. The address for the loop sensor data stream is “sdd.its.washington.edu” at port 9033. This provides the loops data on a 20-second cycle.

Want to see if this connection is for real (and you’re not a programmer)? Open up Netscape on your PC and enter the URL: http://sdd.its.washington.edu: 9033/. Quickly, you will see a lot of data dictionary filling your screen. You won’t be able to see the actual data decoded correctly, but you’ll be able to tell if the port is active or not.