

Open Operational Data Exchange

Draft 1

Kobus Jooste

kobus{at}bixdata.com

May 2005, updated continually

This is an open specification for defining, storing, exchanging and querying operational data. Operational data within this context is defined as any data gathered from computer systems and software such as processor utilization or disk statistics.

1. Introduction

Operational data can be gathered from many sources in the operating system or device drivers. All operational data is unstructured and mostly very low level. There are no type definitions, except for API documentation or source code, and no version information except for what is known about the source.

This makes it hard to store and process operational data in any manageable form, and also makes it difficult to reliably exchange information between different software components, a networked environment or over the internet.

This standard of defining and storing operational data grew out of the desire to effectively source and label data and allow comparison between data from different versions of data sources as well as versions and types of Operating Systems.

2. Goals

This specification attempts to achieve the following goals:

- Specify a method for defining a schema (or data definition) for the different sources of operational data.
- Structure data from an unstructured source within a defined a schema, adding information about the version and environment of the data source as well as timestamp information.
- Make it simple to exchange schemas and data by doing the above in canonical XML, which means it's a subset of XML and all data is encoded in UTF-8.
- Allow easy storing in SQL and OLAP databases.
- Make it possible to query information using XML / XPATH based queries and retrieve data in XML format.
- Ensure future use by using open standards and not controlling schema definitions.

3. Examples of Operational Data

Examples of unstructured operation data:

Linux /proc/stat information of the CPU:

```
cpu 409225533 44570990 397528994 666886241 27352233 2200012
14620262 0
cpu0 202581657 22511115 192908448 341912993 13910596 914826
6452506 0
cpu1 206643876 22059875 204620545 324973247 13441637 1285186
8167755 0
```

The above contains information about the two CPUs of the server, as well as a total of all CPUs.

The first problem is that it is not obvious what any of this information represents. Each version of the Operating System also attaches different meaning to each field, or may remove some fields or add additional fields. The information represented by the same Operating System on different hardware also differs.

Linux /proc/diskstats information about the I/O statistics of each disk device:

```
8 0 sda 33678297 7294384 1791086498 442145271 111220829
222248475 2669331528 2041293131 0 182876914 2484498630
8 1 sda1 414347 6922956 120206855 961654840
8 2 sda2 1027314 8218512 1477197 11817576
8 3 sda3 2782775 35312130 88698706 709589648
8 4 sda4 0 0 0 0
```

The same problems apply here as with CPU data, however in this case there is a relationship, since sda is the parent of sda1-4.

4. Data Definitions and Schemas

A lot of operational data contains a logical hierarchy, even though it is not formally presented in a hierarchy. So it was important to capture this and other structural information and for the schema to show this hierarchy. It is also very important this hierarchy can be preserved, even when stored in a SQL database, or some benefit derived from it when used in an OLAP database.

From gathering and storing operational data for more than a year, it became apparent that the data can be structured into dimensions. Each dimension serves to organize or provide hierarchical structure to the data.

This specification contains the following elements in the schema to present data with:

- **Namespace**; serves to organize data into logical groups such as CPU, Disk, File-System, SMART, TCP and UDP
- **Schema**; within the Disk namespace there may be many types of disks, all with very different information available. Different schemas can exist with a namespace to store information, such as a schema for ATA disks, or a schema for SCSI disks.
- **Version**; each schema has a version associated with it to allow newer versions of schemas or schemas that present drastically different versions of data sources
- **Instance**; most data sources of operational data can provide information on many objects that are all similar. A server may have 2 processors, or 4 disk drives. Each of these is an instance.
- **Key**; at this point the key could serve as the data point (or end point) of the actual data stored. But many devices not only contain different data points for each device, they also have attributes for each data point. This translates into a table of data of columns and rows for each device. A simple example would be the SMART data associated with a disk drive. Each disk has many attributes such as Temperature, Error rate, etc. For each of these attributes there is a multiple data points, such as the raw value, the threshold and the worst value.
- **Field**; are similar to fields in a database. Fields store the data points for each set of data, and examples of fields for CPU are the CPU utilization, CPU idle time, temperature, etc.

Instances represent system entities, devices or any object that has some definition and data associated with it. Each set of data associated with an instance can be seen as a table, where the keys represent rows, and the fields represent columns.

Here is an example of a table representing SMART hard disk data:

Keys	Instance: /hda			
	Fields			
		Raw Value	Worst	Threshold
	CRC Error Rate	0	253	42
	Temperature	24	87	42
Spin Retry	100	100	0	
Seek Error Rate	1	23	0	

Namespace: SMART, Schema: ATA, version: 1

And in this table namespace, schema and version together represent the type of table and data represented.

4.1 Data Definition Sample

Here is a sample definition in XML to represent Memory Information. This data definition represents two schemas, one for common memory information (all operating systems) and one specific for Linux.

```
<DataDefintion>
  <namespace>Memory</namespace>
  <version>1</version>
  <Schema>
    <name>Common</name>
    <Field>
      <name>FreeMemory</name>
      <type>uint32</type>
      <display>Free Memory</display>
      <userdata>Free memory - bytes</userdata>
    </Field>
    <Field>
      <name>TotalMemory</name>
      <type>uint32</type>
      <display>Total Memory</display>
      <userdata>Total memory - bytes</userdata>
    </Field>
    <Instance>
      <name>Onboard</name>
      <display>Onboard Memory</display>
    </Instance>
  </Schema>
  <Schema>
    <name>Linux</name>
    <Field>
      <name>FreeSwap</name>
      <type>uint32</type>
      <display>Free Swap</display>
      <userdata>Free swap - bytes</userdata>
    </Field>
    <Field>
      <name>UsedSwap</name>
      <type>uint32</type>
      <display>Used Swap</display>
      <userdata>Used swap - bytes</userdata>
    </Field>
    <Instance>
      <name>DiskSwap</name>
      <display>Disk Swap</display>
    </Instance>
  </Schema>
</DataDefintion>
```

Each of the <field> entry describes a data point. Note, the only required elements are the name of the field and the data type, <display> and <userdata> are optional to provide more information for human readable presentation of the data, and perhaps formatting or descriptions of the data.

Only one <instance> is present for the Common schema, which is onboard memory. Other types of memory may also be presented, such as flash memory used for booting operating systems, and will just become another <instance>

5. Representing Operational Data

It is quite simple to represent the unstructured data from different data sources. Data is represented in XML and need only conform to the data definition, such as the one provided above for Memory data.

Since all data is time sensitive, a global <timestamp> that represents seconds since 1970 is present in each data section. Many tools can sample data from different sources for the same timestamp; however if this is not possible, different timestamps can be represented by simple creating a section for each timestamp.

Multiple namespaces can be represented in the same XML file or section of data, simply by wrapping each namespace's data in a <Namespace> tag.

```
<Data>
  <timestamp>1143072000</timestamp>
  <Namespace>
    <name>Memory</name>
    <version>1</version>
    <Instance>
      <schema>Common</schema>
      <name>Onboard</name>
      <Key>
        <name></name>
        <Fields>
          <FreeMemory>510016</FreeMemory>
          <TotalMemory>1048040</TotalMemory>
        </Fields>
      </Key>
    </Instance>
    <Instance>
      <schema>Linux</schema>
      <name>DiskSwap</name>
      <Key>
        <name></name>
        <Fields>
          <FreeSwap>10000</FreeSwap>
          <UsedSwap>1024456</UsedSwap>
        </Fields>
      </Key>
    </Instance>
  </Namespace>
</Data>
```

Each object in the namespace can be of a specific type, which is represented by the schema for that type. In this case there is information for the onboard memory as well as

the disk swap space from Linux. In the case of Memory there is no need for the additional level of depth as required by some other data sources, so the Key is simply left empty.

Each one of the <Fields> simply lists the fields for which there is information available. This uses the fieldname as the XML tag, and the value of the field, to be as concise as possible.

6. Exchanging Operational Data

Exchanging operational data between software components can be done by sending the XML representation of the data. The Data Definition only needs to be provided once whenever it becomes available. Data Definitions can not change, and requires new versions of a Data Definition to incorporate modifications.

Since the XML is basically very simple, without attributes or xml namespaces, it becomes trivial to optimize the representation of the XML data in a way more suitable for transfer between networked entities. One possible way of optimizing the size and speed of transfer is through Binary XML.

To exchange operational data between third party applications or through the internet, parties only need to exchange the Data Definition and data. Both can be wrapped in an OODE envelope.

```
<OODE-Envelope>
  <DataDefinition>...</DataDefinition>
  <Data>...</Data>
  <Data>...</Data>
</OODE-Envelope>
```

7. Storing Operational Data in a SQL Database

The schema and Data Definition was specifically designed to allow storing of data in a SQL database. The Data Definition consists of four dimensions of data, and needs to be 'flattened' to be stored in a SQL Database.

The namespace and schema is represented by a table in the SQL database. Tables are created with a naming convention as follow:

Namespace_version_schema

The remaining dimensions for instance, key and fields can be reduced to a SQL database schema as follow:

- Fields are columns in the table, created using the type information from the Data Definition

- The dimensions for instance and key are reduced to reserved columns `_Instance` and `_Key`

In addition to these fields, two reserved columns are created to store the timestamp and an identifier for the source of the data, which usually is the machine identifier or machine name.

- The timestamp is created as a reserved column, `_Timestamp`
- A unique identifier for the data source is stored in a reserved column, `_SourceKey`

Creating the actual tables in SQL is a simple process of transposing the Data Definition to SQL CREATE TABLE statements for each supported database.

Here are two tables in SQL that correspond to the Data Definition for the Memory namespace presented earlier, as well as one row of sample data.

Database Column	Data Type	1 Row of Data
<code>_Timestamp</code>	BigInt(20)	1143072000
<code>_SourceKey</code>	Text	clusternode1
<code>_Instance</code>	Text	OnBoard
<code>_Key</code>	Text	NULL
<code>FreeMemory</code>	Integer	510016
<code>TotalMemory</code>	Integer	1048040

Table: Memory_1_Common

Database Column	Data Type	1 Row Of Data
<code>_Timestamp</code>	BigInt(20)	1143072000
<code>_SourceKey</code>	Text	clusternode1
<code>_Instance</code>	Text	DiskSwap
<code>_Key</code>	Text	NULL
<code>FreeSwap</code>	Integer	10000
<code>UsedSwap</code>	Integer	1024456

Table: Memory_1_Common

8. Querying and Retrieving Data from SQL Database

This specification defines a method for queries in XML to retrieve operational data from XML files or a SQL database. Queries are defined in XML and translated into SQL SELECT statements. The translation from the query defined in XML to SQL SELECT statement is fairly trivial in most scripting languages, Java, C++ or even a template processor such as XSLT. Once the query is performed, the resulting data is output to XML.

Queries consist of one or more `<TableQuery>` entries which specify the namespace, schema, instance, key and field parameters. Each table query can also be limited to include only the latest values, ordered by timestamp, by omitting the `<retrieve>` tag.

```
<Query>
  <Pivot>
    <type>_SourceKey</type>
    <source>*</source>
  </Pivot>
  <TableQuery>
    <namespace>Memory</namespace>
    <version>1</version>
    <schema>Common</schema>
    <instance>*</instance>
    <key>*</key>
    <field>FreeMemory</field>
    <field>TotalMemory</field>
    <retrieve>all</retrieve>
  </TableQuery>
</Query>
```

The resulting data can be pivoted around a specified column in a table, thereby grouping results together based on a common value. `<Pivot>` specifies how the results are grouped together. In this example results are grouped based on the source key, which is the machine identifier. Results can also be pivoted by specific instances, such as an installed hard disk, or specific keys, such as a type of CPU. `<Pivot>` is also used to specify the source keys used in the query; in this example all source keys are used.

The results for the above query will be returned in XML, close to the original representation of the data from the data source, with the addition of a `<Pivot>` that groups results together based on the query input XML.

```
<Result>
  <Pivot>
    <key>clusternode1</key>
    <TableData>
      <namespace>Memory</namespace>
      <version>1</version>
      <schema>Common</schema>
      <Instance>
        <value>OnBoard</value>
        <Key>
          <value></value>
          <timestamp>1143072000</timestamp>
          <Fields>
            <FreeMemory>510016</FreeMemory>
            <TotalMemory>1048040</TotalMemory>
          </Fields>
        </Key>
      </Instance>
    </TableData>
  </Pivot>
</Result>
```

The corresponding SQL for the table query is as follow:

```
SELECT _SourceKey,_Instance,_Timestamp,_Key,_TotalMemory,_FreeMemory
from Memory_1_Common, order by _SourceKey,_Instance,_Timestamp ASC
,_Key
```

9. Future

More complex systems and environments require comparison and analysis of operational data from many different sources. Representing operational data in XML and using published conventions and data definitions (or schemas) for exchanging data, will ensure the correctness and usefulness of operational data. Better data will lead to better tools and a better understanding of problems in the area of systems management.

Published on <http://www.bixdata.com>