What Repeating Groups in Printer MIB Become What Classes in CIM?

RBLandau v0.2 20060925

Question 1: How shall we represent SNMP tables in CIM?

The SNMP Printer MIB and the Printer Semantic Model contain many repeating groups which will be represented in the CIM model in some data structure. There are several possibilities. What structures shall we use to represent what repeating groups?

The repeating group structure in SNMP is simple, at least for the objects in printers: rectangular tables.

fooTable SEQUENCE OF fooTableEntry
fooTableEntry SEQUENCE OF fooRowVector
fooRowVector = tuple (index, prop1, prop2, ...)
(approximately)

The result: a rectangular table with one row per repeating item, where the columns of the row are the properties of the particular instance.

There are (at least) four methods of representing repeating groups in CIM:

1. One collection, row instances, and row associations
2. A series of parallel arrays
3. An array of embedded objects
4. An array of references to row instances

None of these structures is without precedent in the current CIM mode. However, some of them are preferred for various reasons.

Structure 1. One class to represent the row object, and one instance of that class per row of the table. One collection class to represent the table object, and exactly one instance of that class. One class to represent the association of the row as a member of the table, and one instance per row of the table. The index information for a table row is typically stored in the association instance. (This finely-divided structure is like a CODASYL database, if anyone remembers back that far.)

fooTable class derived from Collection
fooVector new class = (index, prop1, prop2, ...)
fooVectorInTable new association class

The result is one fooVector instance per item, plus one fooTable instance, plus one fooVectorInTable instance per item relating Vector instance to table parent.

Structure 2. Parallel array structure, where each array represents a column of the table. One ordered array per property, including the index property.

```
uint16  index[],
string   prop1[],
integer prop2[],
 ...
```

Structure 3. An array of embedded objects, where each embedded object is a row of the table. One class defined for the row object. Another class includes the array of embedded row objects.

```
EmbeddedInstance("WidgetCo_FooRowVector") ...
string FooTable[];
```

Structure 4. An array of references to row objects. One class defined for the row object, and one instance of that object per row. Another class includes the array of references to the row object instances.

```
WidgetCo_FooRowVector  REF  FooTable[];
```

What structure(s) to use? I spoke at length with the vice-chair of the CIM Core Working Group, which approves all schema changes, and he is strongly of the opinion that Structure 1 would be considered the standard way to represent such a structure, and one proposing that structure would probably receive no argument. On the other hand, while all of the other structures have been used in the past, they have been considered controversial recently.

- The default structure in CIM is Structure 1, the row instances associated with a table object.

- If the data is purely declarative or status, read-only to the user, then one might argue for Structure 2. However, an instance of this structure was recently rejected in the new processor model.

- Structures 3 and 4 are rarely used in CIM, and are not currently well-regarded. They have been used in the past for complex storage structures defined by SNIA. Most instances of embedded objects are, I believe, in argument lists of class methods.

Considerations:

- Is the data mutable? Can it change within a power cycle? If data is not mutable, and therefore is read once by the client and cached, then a simple structure is less work for both provider and client.

- Can the user update the data within the table? If single data items can be updated without coordination with other data items, then simple structures, e.g., parallel arrays, are possible. If multiple data items must be updated, then separate row objects are a better structure.

- Is the geometry of the table mutable? Hot-plug of any sort might require additions or deletions of rows. If the geometry of the table changes at runtime, then separate row objects are a better structure.

- Can the user update the geometry of the table? I don't believe that this happens in the printer space.

Repeating groups in Printer MIB and Semantic Model:

Covers
Localizations (xxx not in SM)
StorageRefs (ignore)
DeviceRefs (ignore)
InputTrays
OutputTrays
MarkerSupplies
MarkerColorants
Markers (in SM subsumes Supplies and Colorants)
MediaPath
InputChannels
Interfaces (added in SM)
Finishers (added in SM)
Interpreters
ConsoleDisplayBuffer (xxx not in SM)
ConsoleLights (xxx not in SM)
Consoles (added in SM)
Alerts
Scanners (ignore, added in SM)
VendorSubunits (ignore, added in SM)

Of these, the ConsoleDisplayBuffer seems to be a reasonable standalone vector of strings. In CIM, arrays don't even need a count of rows as a separate property. All the other groups seem that they might have to be treated as CIM_Collections.