EOS WG proposal

Wes Hardaker
<hardaker@tislabs.com>

draft-hardaker-eos-oops-00.txt

2002.Jul.18

Agenda

- Overview: motivations and PDU definitions
- Examples
- Benifits
- Known Issues and Questions

Motivations

- Solve the problems recently enumerated.
- Promote speed-up of configuration use.
- Promote efficient use of SMIng data structures.

Notes about screens to follow

- { = begin BER sequence
- } = end of BER sequence

- Draft recommends TCP.
  - Objects returned will still be smaller than currently returned SNMP objects, but will still be large.

GET replacement: GORP

```
GetObjectRequestPDU {
    request-id      1
    max-objects     0         -- 0 == get all
    skip-objects    0         -- starting with the first
    error-string    ""        -- needed?  There only to match response.
    request-objects ::= {
      {                       -- sic
        Object-OID  (e.g. table)
        { INTEGER:index-number, BER:value }
        { ... }, ...
      }
      { more requests ... }
    }
}
```

GORP RESPONSE

(which is, uh, still GORP...)

```
GetObjectResponsePDU {
    request-id      1
    error-status    0
    error-index     0
    error-string    ""
    request-objects ::= {
     {
       ObjectOID  (e.g. table)
       {     -- begin returned object list
         {   -- returned object 1
             -- [all] indexes for this object
           { { INT:index-number, BER:index-value }
             { INT:index-number, BER:index-value}
             ...
           }
             -- data
           { { INT:component-number, BER:component-value }
             { INT:component-number, BER:component-value }
             ...
           }
         }
       }     -- end object list
     }
```

Example 1: GORP the 5th interface from the ifTable

```
GetObjectRequestPDU {
    request-id      1
    max-objects     0                    -- 0 == get all
    skip-objects    0                    -- starting at first
    error-string    ""
    request-objects ::= {
      {                                  -- sic
        base-object ::= OID:ifTable
        search-list ::= {
          index-number = INTEGER:1       -- ifIndex
          value        = INTEGER:5
        }
      }
   }
}
```

Example 1: response: the 5th interface from the ifTable

```
  GetObjectResponsePDU {
     request-id      1
     error-status    0            -- noError
     error-index     0            -- must be 0 if noError
     error-string    ""
     request-objects ::= {
       {
         object-name ::= OID=ifTable
         object-data ::= {
           {
-- row indexes
             { { index-number     ::= INTEGER: 1,
                 value            ::= INTEGER: 5 } }
-- data
             {
               { component-number ::= INTEGER: 1,
                 value            ::= INTEGER: 5 },
               { component-number ::= INTEGER: 2,
                 value            ::= STRING:  "interface 5" },
               ...
               { component-number ::= INTEGER: 22,
```

Example 2: GORP TCP connections from 10.0.0.1 to www.ietf.org

```
GetObjectRequestPDU {
    request-id      2
    max-objects     2              -- get only 2
    skip-objects    2              -- skipping two, (get the third)
    request-objects ::= {
      {                            -- sic
        tcpConnTable
        {
          { 1, 10.0.0.1 }
          { 3, 4.17.168.6 }
        }
      }
    }
}
```

Example 2: GORP TCP connections from 10.0.0.1 to www.ietf.org

```
GetObjectResponsePDU {
   request-id      2
   error-status    0            -- noError
   error-index     0            -- must be 0 if noError
   request-objects ::= {
     {
       tcpConnTable
       {
         {
           { { 1, 10.0.0.1 },
             { 2, 9876 },
             { 3, 4.17.168.6 },
             { 4, 80 } },       -- web
           { { 1, 5 },          -- 5 = established
             { 2, 10.0.0.1 },
             { 3, 9876 },
             { 4, 4.17.168.6 },
             { 5, 80 } }
         }
         {
           { { 1, 5 },          -- 5 = established
             { 2, 10.0.0.1 },
```

Example 3: GORP of SMIng example

```
VAR ARRAY something {
   INDEX { anInt }
   LEAF anInt {
      SYNTAX Integer32
   } ::= 1                          someBase.1.[1].anInt                    = 42
                                 someBase.1.[1].name[me]                    = me
   ARRAY anArray {                  someBase.1.[1].anArray.[me].type        = 2
      INDEX { name }                someBase.1.[1].anArray.[me].value.string = hello
                                 someBase.1.[1].name[you]                   = you
      LEAF name {                   someBase.1.[1].anArray.[you].type       = 1
         SYNTAX OctetString         someBase.1.[1].anArray.[you].value.int  = 9999
      } ::= 1

      LEAF type {
         SYNTAX Integer32
      } ::= 2

      UNION value {
         LEAF int {
            SYNTAX Integer32
         } ::= 1

         LEAF string {
            SYNTAX OctetString
         } ::= 2
      } ::= 3
   } ::= 2
} ::= { someBase 1 }
```

Example 3: GORP of SMIng example

```
GetObjectRequestPDU {
    request-id      3
    max-objects     0             -- get all
    skip-objects    0             -- skipping none, (get the first)
    request-objects ::= {
      {                           -- sic
        someBase
        {                         -- no indexes, get all
        }
      }
    }
}
```

Example 3: GORP of SMIng example

```
GetObjectResponsePDU {
    request-id      3
    error-status    0           -- noError
    error-index     0           -- must be 0 if noError
    error-string    ""
    request-objects ::= {
      {
        someBase
        {
          {
            { { 1, 1 } },                       index: anInt = 1
            { { 1, 1 },                         anInt = 1
-- array item 1
              { 2, { { { { 1, "me" } },           index: name = "me"
                     { { 1, "me" },             name = "me"
                       { 2, 2 },                type = 2
                       { 3, { 1, "hello" } } } },   value.string = "hello"
-- array item 2
                   { { { 1, "you" } },           index: name = "you"
                     { { 1, "you" },             name = "you"
                       { 2, 2 },                 type = 2
                       { 3, { 2, 42 } } } },     value.int = 2
                 }
              }
            }
          }
        }
      }
    }
  }
  = 166 bytes in BER
```

Example 3: XML-like of SMIng example

```
<GetObjectResponsePDU>
   <request-id>3</request-id>
   <error-status>0</error-status>
   <error-index>0<error-index>
   <response-objects>
     <base-oid>someBase</base-oid>
     <object-list>
       <indexes>
         <anInt>1</anInt>
       </indexes>
       <data>
         <anInt>1</anInt>
         <anArray>
           <indexes>
             <name>me</name>
           </indexes>
           <data>
             <name>me</name>
             <type>1</type>
             <value><string>hello</string></value>
           </data>
           <indexes>
             <name>you</name>
           </indexes>
           <data>
             <name>you</name>
             <type>2</type>
             <value><int>42</int></value>
           </data>
         </anArray>
       </data>
     </object-list>
```

Solves: Get-NEXT/BULK complexity

- Agents return objects as they order them
  - (However, order must be consistent).
- Managers sort the data themselves if need be.
- Selection of data done by "skipping" and simple "==" filtering
  - Get X objects, skip Y
  - Can specify "index1 = this, index3 = that"
    - But, index2, index4, ... unfiltered
    - See the draft for a tcpConnTable example
- Hole traversal solved by BER grouping.

Index parsing and OID length restrictions

- Old index parsing is algorthimic, but no one gets it right.
  - Indexes are encoded using BER instead.
  - OIDs are truncated at the table node.
    - (Achieves compression).
  - You don’t need the MIB to decode the index data.
- OIDs no longer impose a size restrictions on indexes.

Data Conversion

- XML is popular (therefore it must be good right?)
- Trival conversion to/from XML:
  - Replace every sequence tag with a <something>
  - Replace every sequence end with a </something>
- A naming conversion scheme not defined.
  - Must define the word "something" above.

SET Operations

- Data PUSH operations not in the draft.
  - (they’re on a piece of paper on my desk at home)
- Functionally, they’ll contain:
  - Textual error descriptions
  - Do-All vs Do-Any operations (transaction fixes)
  - Create/Replace/(delete?) row-ops

Issues:

- Filtering/Searching on non-indexes?
- Filtering on other expression types.
  - INTEGER: <, >, <=, >=
  - STRING: prefix, regexp, ...
  - make them optional?
    - server simply returns more data if op not supported?

Issues:

- VACM Access Control maybe broken for fine-grained control
  - Can still provide access to the table granularity level
  - To provide fine-grained access:
    - For VACM, need to re-encode indexes into OIDs (sigh)
    - or use something else

Issues:

- Thoughts?
- Desires?
- Popularity?