SNMP - LTM v9.x MIB Navigation

Deb Allen, 2007-27-06

I had the opportunity to dissect the new LTM SNMP v9 MIB back when 9.0 first rolled. Here's a quick overview of how to navigate in its sometimes murky waters.

The LTM v9 MIB is considerably different from the BIG-IP 4.x MIB, and some of the new OIDs are not exactly intuitive, so I built a spreadsheet breaking out the MIB into tabs for each feature (LTMv9.0_MIB_deconstructed.zip) and some navigation techniques that have been really helpful in finding specific object OIDs. (Note: The spreadsheet only touches the most commonly used parts, and not all current features existed then.)

The spreadsheet correlates OIDs with human-readable names for the permanent branches of the MIB, so you can search by either one. That should save you some time, because it actually can be surprisingly difficult to find what you're looking for from the command line.

I've included in the spreasheet only the base OIDs, the ones that are permanent in the MIB. Wherever you see a "*", that means there are OIDs below that enumerate automatically for each object added. You'll want to check out the "Interesting Stats" tab in the spreadsheet for the specific OIDs you'll most likely be interested in. The "tree-relationships" tab shows the MIB-2 relationships for the non-F5 system MIBs, in case cross-referencing is required. The other tabs contain related MIBs or an entire MIB section for an LTM feature or feature set, and the purple highlighted rows are the ones that populate the "Interesting Stats" tab.

But perusing the spreadsheet will only get you so far. Once you've found the base OID, you may need to get a specific numeric OID for a single object.

Leaf objects are enumerated under the base branch in name-specific OIDs constructed using the ASCII-encoding decimal values for each character in the VS name.

For instance, the OID below is the base "VS client bytes out" OID:

```
.1.3.6.1.4.1.3375.2.2.10.2.3.1.9
```

To find an exact OID, you can figure out the ASCII equivalent of the VS/object name using your secret ASCII decoder ring

```
118.115.95.116.101.115.116 = "vs_test"
```

For OIDs related to objects with variable length string names or IP addresses, you'll also have to determine the "prefix" for the OID leaf. For string named objects, just count the number of characters in the object name (**7** in this case). For IP address objects, the prefix will be either 4 for IPv4 or 6 for IPv6.

then construct the OID from these parts:

```
Base branch:                    .1.3.6.1.4.1.3375.2.2.10.2.3.1.9
Leaf prefix (# chars or version)   .7
Object name (ASCII):            .118.115.95.116.101.115.116
```

so the "VS client bytes out" OID for a specific VS named "vs_test" would be:

```
.1.3.6.1.4.1.3375.2.2.10.2.3.1.9.7.118.115.95.116.101.115.116
```

Alternatively, you can walk down the MIB branch to find the sepcific OID you want like this:

1. snmpwalk one or 2 levels above the base OID with -On output flag, grepping for object name:

```
snmpwalk -c public -v2c localhost -On .1.3.6.1.4.1.3375.2.2.10.2.3.1| grep
test
```

which returns:

```
        .1.3.6.1.4.1.3375.2.2.10.2.3.1.1.7.118.115.95.116.101.115.116 = STRING:
vs_test
```

2. examine output for encoding of object name -- will be end of numeric OID:

```
        118.115.95.116.101.115.116 = "vs_test"
```

3. snmpwalk the base OID with -On flag, grepping for encoded name, and examine output for the numeric OID:

```
        snmpwalk -c public -v2c localhost -On .1.3.6.1.4.1.3375.2.2.10.2.3.1.9 |
grep 118.115.95.116.101.115.116
```

   which returns:

```
        .1.3.6.1.4.1.3375.2.2.10.2.3.1.9.7.118.115.95.116.101.115.116 = Counter64:
0
```

Note the ".**7**" sandwiched between the base OID and the encoded VS name.

```
        .1.3.6.1.4.1.3375.2.2.10.2.3.1.9.7.118.115.95.116.101.115.116 = Counter64:
0
```

It's that automagically inserted decimal counter of the total number of characters in the object name that I mentioned previously.