Data Queries

Data queries are not a replacement for data input methods in Cacti. Instead they provide an easy way to query, or list data based upon an index, making the data easier to graph. The most common use of a data query within Cacti is to retrieve a list of network interfaces via SNMP. If you want to graph the traffic of a network interface, first Cacti must retrieve a list of interfaces on the host. Second, Cacti can use that information to create the necessary graphs and data sources. Data queries are only concerned with the first step of the process, that is obtaining a list of network interfaces and not creating the graphs/data sources for them. While listing network interfaces is a common use for data queries, they also have other uses such as listing partitions, processors, or even cards in a router.

One requirement for any data query in Cacti, is that it has some unique value that defines each row in the list. This concept follows that of a 'primary key' in SQL, and makes sure that each row in the list can be uniquely referenced. Examples of these index values are 'ifIndex' for SNMP network interfaces or the device name for partitions.

There are two types of data queries that you will see referred to throughout Cacti. They are script queries and SNMP queries. Script and SNMP queries are virtually identical in their functionality and only differ in how they obtain their information. A script query will call an external command or script and an SNMP query will make an SNMP call to retrieve a list of data.

All data queries have two parts, the XML file and the definition within Cacti. An XML file must be created for each query, that defines where each piece of information is and how to retrieve it. This could be thought of as the actual query. The second part is a definition within Cacti, which tells Cacti where to find the XML file and associates the data query with one or more graph templates.

Creating a Data Query

Once you have created the XML file that defines your data query, you must add the data query within Cacti. To do this you must click on Data Queries under the Data Gathering heading, and select Add. You will be prompted for some basic information about the data query, described in more detail below.

Table 12-1. Field Description: Data Queries

Name	Description
Name	Give the data query a name that you will use to identify it. This name will be used throughout Cacti when presented with a list of data queries.
Description	(Optional) Enter a more detailed description of the data query including the information it queries or additional requirements.
XML Path	Fill in the full path to the XML file that defines this query. You can optionally use the <path_cacti> variable that will be substituted with the full path to Cacti. On the next screen, Cacti will check to make sure that it can find the

	XML file.
Data Input Method	This is how you tell Cacti to handle the data it receives from the data query. Typically, you will select “Get SNMP Data (Indexed)” for an SNMP query and “Get Script Data (Indexed)” for a script query.

When you are finished filling in all necessary fields, click the Create button to continue. You will be redirected back to the same page, but this time with some additional information to fill in. If you receive a red warning that says 'XML File Does Not Exist', correct the value specified in the 'XML Path' field.

Associated Graph Templates

Every data query must have at least one graph template associated with it, and possibly more depending on the number of output fields specified in the XML file. This is where you get to choose what kind of graphs to generate from this query. For instance, the interface data query has multiple graph template associations, used to graph traffic, errors, or packets. To add a new graph template association, simply click Add at the right of the Associated Graph Templates box. You will be presented with a few fields to fill in:

Table 12-2. Field Description: Associated Graph Templates

Name	Description
Name	Give a name describing what kind of data you are trying to represent or graph. When the user creates a graph using this data query, they will see a list of graph template associations that they will have to choose from.
Graph Template	Choose the actual graph template that you want to make the association with.

When you are finished filling in these fields, click the Create button. You will be redirected back to the same page with some additional information to fill in. Cacti will make a list of each data template referenced to in your selected graph template and display them under the Associated Data Templates box. For each data source item listed, you must selected the data query output field that corresponds with it. **Do not forget to check the checkbox to the right of each selection, or your settings will not be saved.**

The Suggested Values box gives you a way to control field values of data sources and graphs created using this data query. If you specify multiple suggested values for the same field, Cacti will evaluate them in order which you can control using the up or down arrow icons. For more information about valid field names and variables, read the section on suggested values.

When you are finished filling in all necessary fields on this form, click the Save button to return to the data queries edit screen. Repeat the steps under this heading as many times as necessary to represent all data in your XML file. When you are finished with this, you should be ready to start adding your data query to hosts.

SNMP Query XML Syntax

```
<query>
   <name>Get SNMP Interfaces</name>
   <description>Queries a host for a list of monitorable interfaces</description>
   <oid_uptime>.1.3.x.x.x</oid_uptime>
   <oid_index>.1.3.6.1.2.1.2.2.1.1</oid_index>
   <oid_index_parse>OID/REGEXP:.*\.([0-9]{1,3}\.[0-9]{1,3})$</oid_index_parse>
   <oid_num_indexes>.1.3.6.1.2.1.2.1.0</oid_num_indexes>
   <index_order>ifDescr:ifName:ifIndex</index_order>
   <index_order_type>numeric</index_order_type>
   <index_title_format>|chosen_order_field|</index_title_format>
   <fields>
      <ifIndex>
         <name>Index</name>
         <method>walk</method>
         <source>value</source>
         <direction>input</direction>
         <oid>.1.3.6.1.2.1.2.2.1.1</oid>
      </ifIndex>
   </fields>
</query>
```

Table 12-3. SNMP Query XML Field Reference

Field	Description
query→name	(Optional) You can enter a “friendly name” for the SNMP query here. It will not be used by Cacti, and is for identification only.
query→description	(Optional) You can enter a description for the SNMP query here. It will not be used by Cacti, and is for identification only.
query→oid_uptime	(Optional, new with 0.8.7): If you have another OID that contains timetics, say for example a Java VM. Then, you can create a data query that specifies an alternate Uptime OID. To implement this for a data query, simply add the oid_uptime XML parameter to your XML file. Then, if you select your re-index method to be Uptime Goes Backaward, Cacti will use that OID to detect whether it is time to re-index the host instead of the standard snmp OID for uptime.
query→oid_index	Every SNMP query must have an OID that represents the index values for the query when walked. As described above, any data query in Cacti must contain a field that uniquely identifies each row returned by the query. In the example above, the oid_index points to the OID of ifIndex in the interface MIB. **Note:** Starting with version 0.8.6c, Cacti is able to parse unique indexes from the OID itself. While the regular expression used for parsing the value from the OID is defined below, you must still specify an OID that can be walked by Cacti in order to obtain the list of OID's. Any OID defined for one of your input fields should work in this case. The values returned from the snmpwalk walk will be completely disregarded.

query→oid_index_parse	(Optional) This field should only be used if you are trying to parse the unique index from the OID itself. If this field is defined, to obtain a list of indexes, Cacti walks the OID provided in the oid_index field above. It then applies the regular expression provided in this field to the list of OID's that are returned. The matched substrings that remain become the list of indexes for this SNMP query.
query→oid_num_indexes	(Optional) [Re-Index Method = **Index Count Changed**] An OID that can be queried to determine the **total number of available indexes**. If specified, this will be used to determine when to automatically recache this SNMP query when it is attached to a device. [Re-Index Method = **Index Value Changed** (new since 0.8.8)] In this case, the <oid_num_index> **value** is taken to determine if a re-index is required.
query→index_order	(Optional) As of version 0.8.6, Cacti will attempt to find the best field to index off of based on whether each row in the query is unique and non-null. If specified, Cacti will perform this check on the fields listed here in the order specified. Only input fields can be specified and multiple fields should be delimited with a colon.
query→index_order_type	(Optional) For sorting purposes, specify whether the index is numeric or alphanumeric. **numeric**: The indexes in this SNMP query are to be sorted numerically (ie. 1,2,3,10,20,31) **alphabetic**: The indexes in this SNMP query are to be sorted alphabetically (1,10,2,20,3,31). **natural**: Introduced to handle IP address style data. Given the dotted-quad representation of a network address as a string, returns an integer that represents the numeric value of the address to determine the sort order.
query→index_title_format	(Optional) Specify the title format to use when representing an index to the user. Any input field name can be used as a variable if enclosed in pipes (|). The variable "|chosen_order_field|" will be substituted with the field chosen by Cacti to index off of (see index_order above).
query→fields	Each field contained within the SNMP query must be defined under this tag.
query→fields→ifIndex	Each defined field in the SNMP query must have a unique name given to it. Do not use spaces or any non-alphanumeric characters, this name must be identifiable within Cacti.
query→fields→ifIndex→name	Here you can specify a “friendly name” for the field. This name will be used by Cacti to help the user identify this field.

query→fields→ifIndex→method	Tell Cacti how you want it to gather SNMP information for this field. **get**: The 'get' method obtains a list of indexes and does an snmpget for each index of the OID specified for this field. **walk**: The 'walk' method does a walk of the OID specified for this field. Both methods will return the same values, even though the 'walk' method is typically more efficient.
query→fields→ifIndex→source	When Cacti obtains a list for this field, you need to tell it how to derive its value for each row. **value**: The 'value' option simply returns the result of the snmpget for each row. **OID/REGEXP**:(regexp_match): The 'OID/REGEXP:(regexp_match)' can be used when you need to use a POSIX-based regular expression to derive the value from the OID. The most common example of this is to retreive the IP address of an interface, and can be seen in the 'interface.xml' file. **VALUE/REGEXP**:(regexp_match): The 'OID/REGEXP:(regexp_match)' option can be used to parse the value based on a regular expression, returning the first match.”index”: Simply use the value of the index for this row as the value. If the index is being parsed from the OID using the oid_index_parse field, you must specify “index” here.
query→fields→ifIndex→direction	**input**: Input values are the “known” values that you will use to derive the output values, this is where the “query” part of SNMP query comes in. When you create a graph based on an SNMP query, Cacti will prompt you to choose the input value to base the graph on. **output**: Output values are “unknown” values that are returned from the script. An SNMP query may return multiple statistics for a single index. For instance, a single interface could return bytes/sec in, errors, packets/sec, etc.A rule of thumb is that input fields contain semi-static data that is not graphable, while the output fields contain the data that will be graphed.
query→fields→ifIndex→oid	You must specify the actual OID that corresponds with the field. Each value for this field can be obtained by doing an snmpget on 'oid.(each)snmpindex'.
query→fields→ <field> → oid_suffix	(Optional, new with 0.8.7e) In case you have to append a specific suffix to the OID for your *<field>*, specify that suffix here. The complete OID for that *<field>* then reads: OID.<index>.<oid_suffix>
query→fields→ <field> → rewrite_index	(Optional, new with 0.8.8) rewrite_index overrides default way to build index of field for SNMP GET query. When no rewrite_index value is specified, Cacti uses the standard query-defined index *<oid_index>*. As soon as *<rewrite_index>* is among properties of input

field with *method=get,* Cacti will evaluate *<rewrite_index>* as a clue to build index.
Let's assume that original index was *$origindex*
There are two kind of tokens:
|query_XXX| will be evaluated as value of XXX with index $origindex
|index| == $origindex
So

```
<rewrite_index>|query_ifName|.33.|index|.|index|.2</rewrite_index>
```

will turn index into

```
{field_value['ifName'][$origindex]}.33.$origindex.$origindex.2
```

rewrite_index is quite useful when some query field is actually key(index) of some other SNMP table. Fields that are used in *<rewrite_index>* should be placed before this field in XML in order to be able to translate |query_XXX| into actual data. Simple example:

```
### inventory table
# index
X.1.1 = 1
X.1.2 = 2
X.1.3 = 3
# item name
X.2.1 = "1000BaseSX SFP"
X.2.2 = "1000BaseSX SFP"
X.2.3 = "1000BaseSX SFP"
# ifIndex
X.3.1 = 3002
X.3.2 = 3003
X.3.3 = 3004

### interface table
# index
Y.1.3002 = 3002
Y.1.3003 = 3003
Y.1.3004 = 3004
# name
Y.2.3002 = "Gi0/1"
Y.2.3003 = "Gi0/1"
Y.2.3004 = "Gi0/1"
```

XML:

```
<oid_index>X.1</oid_index>
<invname>
  <method>walk</method>
  <oid>X.2</oid>
</invname>
<invPifIndex>
  <method>get</method>
  <oid>X.3</oid>
</invPifIndex>
<ifName>
  <method>walk</method>
  <oid>Y.2</oid>
  <rewrite_index>|query_invPifIndex|</rewrite_index>
</ifName>
```

As a result each invname will be associated with ifName

query→fields→ <field> → rewrite_value

(Optional, new with 0.8.8)
Serialized translation map for SNMP return values.
There are cases, where you might want to have SNMP ENUM values printed in a more suitable way. An example is the use of numeric representation of ifOperStatus in the interface MIB.
IF-MIB defines

```
ifOperStatus OBJECT-TYPE
    SYNTAX  INTEGER {
                up(1),        -- ready to pass packets
                down(2),
                testing(3),   -- in some test mode
                unknown(4),   -- status can not be determined
                              -- for some reason.
                dormant(5),
                notPresent(6),    -- some component is missing
                lowerLayerDown(7) -- down due to state of
                                  -- lower-layer interface(s)
            }
```

which translates into a PHP array like this

```
$map = Array
(
    "REGEXPNC:^.*(up|1).*$"             => "Up",
    "REGEXPNC:^.*(down|2).*$"           => "Down",
    "REGEXPNC:^.*(testing|3).*$"        => "Testing",
    "REGEXPNC:^.*(unknown|4).*$"        => "Unknown",
    "REGEXPNC:^.*(dormant|5).*$"        => "Dormant",
    "REGEXPNC:^.*(notPresent|6).*$"     => "notPresent",
    "REGEXPNC:^.*(lowerLayerDown|7).*$" => "lowerLayerDown",
);
```

For use with <rewrite_value>, we need the serialize($map) form of this array (split for sake of readability)

	`a:7:{ s:21:"REGEXPNC:^.*(up|1).*$";s:2:"Up"; s:23:"REGEXPNC:^.*(down\|2).*$";s:4:"Down"; s:26:"REGEXPNC:^.*(testing|3).*$";s:7:"Testing"; s:26:"REGEXPNC:^.*(unknown\|4).*$";s:7:"Unknown"; s:26:"REGEXPNC:^.*(dormant|5).*$";s:7:"Dormant"; s:29:"REGEXPNC:^.*(notPresent\|6).*$";s:10:"notPresent"; s:33:"REGEXPNC:^.*(lowerLayerDown|7).*$";s:14:"lowerLayerDown";}`

Script Query XML Syntax

```
<query>
   <name>Get Unix Mounted Partitions</name>
   <description>Queries a list of mounted partitions on a unix-based host with the 'df' command.</description>
   <script_path>perl |path_cacti|/scripts/query_unix_partitions.pl</script_path>
   <arg_index>index</arg_index>
   <arg_query>query</arg_query>
   <arg_get>get</arg_get>
   <arg_num_indexes>num_indexes</arg_num_indexes>
   <output_delimeter>:</output_delimeter>
   <index_order>dskDevice:dskMount</index_order>
   <index_order_type>alphabetic</index_order_type>
   <index_title_format>|chosen_order_field|</index_title_format>
   <fields>
      <dskDevice>
         <name>Device Name</name>
         <direction>input</direction>
         <query_name>device</query_name>
      </dskDevice>
   </fields>
</query>
```

Table 12-4. Script Query XML Field Reference

Field	Description
query→name	(Optional) You can enter a “friendly name” for the script query here. It will not be used by Cacti, and is for identification only.
query→description	(Optional) You can enter a description for the script query here. It will not be used by Cacti, and is for identification only.
query→script_path	Enter the complete path to the script or executable that is going to handle your script query. When in doubt, specify the pull path to all binaries referenced in this path, the query may not execute otherwise.
query→arg_index	Enter the argument that is to be passed to the script to retrieve a list of indexes.
query→arg_query	Enter the argument that is to be passed to the script to retrieve a list of values given a field name.
query→arg_get	Enter the argument that is to be passed to the script to retrieve a single value given a field name and index value.

query→arg_num_indexes	(Optional) [Re-Index Method = **Index Count Changed**] Enter the argument that is to be passed to the script to determine the **total number of available indexes**. If specified, this will be used to determine when to automatically recache this script query when it is attached to a device. [Re-Index Method = **Index Value Changed**] In this case, the <arg_num_index> **value** is taken to determine if a re-index is required.
query→output_delimeter	Enter the one character delimiter that will be used to separate output values. This is only used when you “query” the script in which case it outputs 'index(delimiter)value'.
query→index_order	As of version 0.8.6, Cacti will attempt to find the best field to index off of based on whether each row in the query is unique and non-null. If specified, Cacti will perform this check on the fields listed here in the order specified. Only input fields can be specified and multiple fields should be delimited with a comma.
query→index_order_type	For sorting purposes, specify whether the index is numeric or alphanumeric.”numeric”: The indexes in this script query are to be sorted numerically (ie. 1,2,3,10,20,31)“alphabetic”: The indexes in this script query are to be sorted alphabetically (1,10,2,20,3,31).
query→index_title_format	Specify the title format to use when representing an index to the user. Any input field name can be used as a variable if enclosed in pipes (|). The variable "|chosen_order_field|" will be substituted with the field chosen by Cacti to index off of (see index_order above).
query→fields	Each field contained within the script query must be defined under this tag.
query→fields→dskDevice	Each defined field in the script query must have a unique name given to it. Do not use spaces or any non-alphanumeric characters, this name must be identifiable within Cacti.
query→fields→dskDevice→name	Here you can specify a “friendly name” for the field. This name will be used by Cacti to help the user identify this field.
query→fields→dskDevice→direction	“input”: Input values are the “known” values that you will use to derive the output values, this is where the “query” part of script query comes in. When you create a graph based on a script query, Cacti will prompt you to choose the input value to base the graph on.”output”: Output values are “unknown” values that are returned from the script. A script query may return

	multiple statistics for a single index. For instance, a single partition could return free disk space, total disk space, fragmentation percentage, etc.A rule of thumb is that input fields contain semi-static data that is not graphable, while the output fields contain the data that will be graphed.
query→fields→dskDevice→query_name	Enter the name that Cacti must use when asking the script for information about this field. For instance, the following should return values: '(script_name) query (query_name)'.

PHP Script Server manual SNMP Data Queries