mod_odbc_query

About

This module can be used for doing ODBC queries as an APP from the dialplan or through the API interface.

Compiling/Enabling

1. You can find the source in the git contrib repository in ledr/c/mod_odbc_query.
2. make/gmake the mod
3. make install
4. copy the .xml config file cp odbc_query.conf.xml /usr/local/freeswitch/conf/autoload_configs/
5. Set your ODBC access in the xml
6. Run "load mod_odbc_query" in the CLI and add it to your auto-load modules list in conf/autoload_configs/modules.conf.xml

v1.4 specific changes

Due to a function name change, the module doesn't compile cleanly with the v1.4.beta or master branches (as of 04.04.2014). Edit: Patch updated (15.04.2014), because of another function change. The following patch resolves the problem:

Expand source

```
diff --git a/ledr/c/mod_odbc_query/mod_odbc_query.c
b/ledr/c/mod_odbc_query/mod_odbc_query.c
index 7d802b2..2f4d3e1 100644
--- a/ledr/c/mod_odbc_query/mod_odbc_query.c
+++ b/ledr/c/mod_odbc_query/mod_odbc_query.c
@@ -117,10 +117,10 @@ static switch_status_t do_config(switch_bool_t reload)
}

/* empty the globals.queries hash */
- for (hi = switch_hash_first(NULL, globals.queries); hi;) {
- switch_hash_this(hi, &key, NULL, &val);
+ for (hi = switch_core_hash_first(globals.queries); hi;) {
+ switch_core_hash_this(hi, &key, NULL, &val);
query = (char *) val;
- hi = switch_hash_next(hi);
+ hi = switch_core_hash_next(&hi);
switch_safe_free(query);
switch_core_hash_delete(globals.queries, (char *) key);
}
@@ -466,7 +466,7 @@ SWITCH_MODULE_LOAD_FUNCTION(mod_odbc_query_load)
switch_mutex_init(&globals.mutex, SWITCH_MUTEX_NESTED, globals.pool);

/* allocate the queries hash */
- if (switch_core_hash_init(&globals.queries, globals.pool) != SWITCH_STATUS_SUCCESS)
{
+ if (switch_core_hash_init(&globals.queries) != SWITCH_STATUS_SUCCESS) {
switch_log_printf(SWITCH_CHANNEL_LOG, SWITCH_LOG_ERROR, "Error initializing the
queries hash\n");
return SWITCH_STATUS_GENERR;
}
@@ -512,10 +512,10 @@ SWITCH_MODULE_SHUTDOWN_FUNCTION(mod_odbc_query_shutdown)
switch_event_unbind_callback(reload_event_handler);

switch_mutex_lock(globals.mutex);
- for (hi = switch_hash_first(NULL, globals.queries); hi;) {
- switch_hash_this(hi, &key, NULL, &val);
+ for (hi = switch_core_hash_first(globals.queries); hi;) {
+ switch_core_hash_this(hi, &key, NULL, &val);
query = (char *) val;
- hi = switch_hash_next(hi);
+ hi = switch_core_hash_next(&hi);
switch_safe_free(query);
switch_core_hash_delete(globals.queries, (char *) key);
}
```

APP

When using odbc_query as an APP you can call it from the dialplan. Returned rows will then be stored as channel variables for later use.

Usage

Create an "odbc_query.conf.xml" file in "autoload_configs", that at least contains an "odbc-dsn":

```
<configuration name="odbc_query.conf" description="ODBC Query Module">
  <settings>
    <param name="odbc-dsn" value="freeswitch:freeswitch:secret"/>
  </settings>
</configuration>
```

Load ''mod_odbc_query'' in your ''modules.conf.xml'':

```
<load module="mod_odbc_query"/>
```

Then you can do queries directly from your xml dialplan:

```
<action application="odbc_query" data="SELECT some_column_name AS
target_channel_variable_name FROM some_table_name WHERE 1;"/>
```

Or,

```
 <action application="odbc_query" data="'''my-query'''"/>
```

The module simply checks whether the data attr contains a space. If it does, then that field will be seen as an SQL query, otherwise it will be seen as a query 'name' which then has to be defined in the modules configuration in a <queries> section like this:

```
<queries>
  <query name="'''my-query'''" value="SELECT some_column_name AS
target_channel_variable_name FROM some_table_name WHERE 1;"/>
</queries>
```

The module will do the query and store each returned column name as channel variable name together with its corresponding value.

Another feature is, that if only two columns are returned, which have the column names "name" and "value", then the channel variables will be set according to them. This way you can have the query return multiple rows with different channel variables. If the query returns something else than column-names "name" and "value" and it returns multiple rows, then the channel variables will be overwritten with each iteration of the rows - which is probably useless.

The query may contain '"${blah}"' variables that will be "'expanded"' from "'channel variables"' before the query is performed.

This application may be run inline from the XML dialplan.

example foo / bar

Query: ""SELECT foo, bar FROM some_table;""

returns:

```
foo    bar
----------
a      b
c      d
```

then the channel variables that will be set are:

```
foo=c
bar=d
```

example name / value

Query: ""SELECT foo AS name, bar AS value FROM some_table;""

returns:

```
name    value
------------
a       b
c       d
```

then the channel variables that will be set are:

```
a=b
c=d
```

So, the first example should only be used when you know that only zero or one row will be returned, and second one if you know zero or more rows will be returned.

If zero rows are returned (in either foo/bar or name/value case) then no channel variables will be set, overwritten or deleted.

example find gateway and bridge to it

> For selecting gateways, you should probably use mod_lcr, this is just for explanation's sake.

For example, you have a table named gateways:

```
id    condition    gateway
-----------------------------------------
1     foo          sip.provider1.com
2     bar          sip.provider2.com
```

Then in your odbc_query.conf.xml:

```
<configuration name="odbc_query.conf">
  <settings>
    <param name="odbc-dsn" value="db:user:pass"/>
  </settings>

  <queries>
    <query name="my_query" value="SELECT gateway FROM gateways WHERE condition =
${what};"/>
  </queries>
</configuration>
```

Then in your xml dialplan, do:

```
 <include>
  <context name="my_context">
    <extension name="lookup_the_gateway" continue="true">
      <condition>
        <action application="set" inline="true" data="what=bar"/>
        <action application="odbc_query" inline="true" data="my_query"/> <!-- now
${gateway} will be 'sip.provider2.com' -->
      </condition>
    </extension>
    <extension name="route_to_gateway">
      <condition field="${gateway}" expression="^.+$"/>
      <condition field="destination_number" expression="^(\d+)$">
        <action application="bridge" data="sofia/some_profile/$1@${gateway}"/>
      </condition>
    </extension>
  </context>
</include>
```

API

The API interface is accessable from several places, for easy testing use it from your fs_cli console.

Usage

```
odbc_query [txt|tab|xml] [db:user:pass] <SELECT * FROM foo WHERE true;>
```

- If you omit the first argument (formatting) then the default 'txt' will be used.
- If you omit the second argument (odbc dsn) then the default odbc-dsn as set in the odbc_query.conf.xml will be used.
- The third argument (the query itself) is mandatory.

As stated above, the formatting can be txt, tab or xml. Here are examples of their output:

format txt

```
fscli> odbc_query txt select 1 as foo, 2 as bar

foo : 1
bar : 2

Got 1 rows returned in 1 ms.
```

format tab

```
fscli> odbc_query tab select 1 as foo, 2 as bar

foo          bar
=======================================
1            2

Got 1 rows returned in 1 ms.
```

format xml

```
fscli> odbc_query xml select 1 as foo, 2 as bar

<result>
  <rows>
    <row>
      <column name="foo" value="1"/>
      <column name="bar" value="2"/>
    </row>
  </rows>
  <meta>
    <error></error>
    <rowcount>1</rowcount>
    <elapsed_ms>1</elapsed_ms>
  </meta>
</result>
```