

Contents

- [1 Introduction](#)
- [2 Compiling/Enabling](#)
- [3 APP](#)
 - ◆ [3.1 example foo / bar](#)
 - ◆ [3.2 example name / value](#)
 - ◆ [3.3 example find gateway and bridge to it](#)
- [4 API](#)
 - ◆ [4.1 format txt](#)
 - ◆ [4.2 format tab](#)
 - ◆ [4.3 format xml](#)

Introduction

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`

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`:

Mod_odbc_query

```
<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

Note: 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:

example name / value

Mod_odbc_query

```
<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
      </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 accessible 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>
```